Final Assessment

Declaration

AI-supported/AI integrated use is permitted in this assessment. I acknowledge the following uses of GenAI tools in this assessment:

- I have used GenAI tools to assist with research or gathering information.
- I have used GenAI tools to help me understand key theories and concepts.
- I have used GenAI tools to identify trends and themes as part of my data analysis.

I declare that I have referenced use of GenAI outputs within my assessment in line with the University referencing guidelines. The GenAI used is ChatGPT only.

Introduction

Spotify, the Swedish audio streaming and media service provider houses over 100 million songs, over 6 million podcasts, and roughly 350,000 audiobooks. The company has over 626 million monthly active users, including 246 million subscribers.

Spotify utilises a freemium revenue model that offers a basic, advertised service for free which can then be upgraded to a premium service for a subscription fee. As of October 2024, the company's market cap is $75.95B (NYSE: SPOT) (*Google Finance*, 2024).

Artists such as 'The Weeknd', 'Taylor Swift', 'Billie Eilish' and 'Coldplay' have roughly 100 million monthly listeners. For each million streams of a song on Spotify, an Artist is expected to make between $1,000-$8000 USD. Taylor Swift is the all-time highest paid Spotify Artist with her total earnings at over $327 million.

The following analysis will aim to understand:

- "What specific musical attributes—such as danceability, BPM, valence, energy, and liveliness—most significantly contribute to the streaming success and chart positions of songs on Spotify?".

- Is there a correlation between the danceability, energy or valence of a song and the number of streams that it has?

- **Hypothesis statement** – does a Hierarchical Model help us understand the relationship between musical attributes and song performance on Spotify.

    - **Null Hypothesis** – The inclusion of artist-level variability does not significantly improve the model's ability to explain the relationship between musical attributes and streams.
    - **Alternate Hypothesis** – The inclusion of artist-level variability (random effects) does significantly improve the model's ability to explain the relationship between musical attributes and streams.

Objectives

The purpose of this analysis is to identify trends or patterns that can explain the success of the top performing songs on Spotify. Through assessing key common musical attributes across the top-rated songs on Spotify, I will use a series of statistical techniques and data visualisation methods in R to uncover how each characteristic relates to song popularity. My process first involves cleaning and wrangling the data where I transformed and standardised variables in order to ensure consistency across each observation. After this, I will utilise packages such as **'dplyr'** for data manipulation, **'ggplot2'** and **'plotly'** for creating informative visualisations and **'kableExtra'** to format and present tables in a polished manner.

After an initial correlation analysis to determine preliminary associations between the key musical variables, I will use both linear modelling and hierarchical modelling. The linear modelling will explore general trends across the dataset, while hierarchical modelling will account for potential variability among individual artists. This will allow me to assess the impact of song attributes at both an overall level and within each artist's unique context. Through interpreting the statistical values obtained from this analysis, inferences will be made to answer the questions laid out above and conclude the hypothesis.

Data

The data used in this analysis was taken from the website 'kaggle.com' which houses thousands of large real world data sets. I selected 'Spotify Most Streamed Songs' and downloaded the data initially as a 'csv' file with 945 rows and 25 columns. After uploading this data into R, I had to manipulate and wrangle the dataset in order to adapt parameters and remove missing data. First, using both **'dplyr'** and **'tidyverse'**, I identified the unique values and looked to see if there was any missing data throughout the 'streams' column using **'sum(is.na())'** function. This returned a single missing data value which I omitted using **'na.omit'.**

After this, I then ensured that the 'danceability_%' and 'streams' columns in the dataset are in a numeric format which would be necessary for computing statistical models and correlations. The code here also removed any commas from the 'streams' column using **'gsub(", ", "", ...)'**, converting the cleaned text to a numeric format since R would read the large numbers as text otherwise. I then updated the 'artists' column so that I could extract only the first artist from each entry and store it in a new column called 'first_artist'. I did this using **'strsplit()'** and **'sapply()'**. I was then able to calculate the number of unique first artists using **'summarise()', 'n_distinct(first_artist)'** and **'pull(unique_first_artist)'**. This outputted a result of 409 unique artists within the dataset.

A moving average was also applied to smooth out the variations in the BPM (beats per minute) variable, providing a clearer view of BPM trends across songs. To do this, I used the **'zoo'** package to calculate a 10-song moving average added in as a new column called 'moving_avg_bpm', which allows for easier analysis of the temporal trends across top-performing songs.

Exploratory Data Analysis and Results

To begin our exploratory data analysis (EDA), I first created a basic summary statistics table to gain an understanding of the key metrics used to describe the songs in the dataset. Figure 1 below shows this table created using the **'kableExtra'** package which allowed me to provide additional styling and customisation options to my table, making it clear and readable. This package also supports HTML and LaTeX outputs, ensuring the table can integrate smoothly with my report's style.

Summary Statistics for Key Metrics

Metric	Mean	Median	Standard Deviation	Min	Max	Unique Artists
BPM	122.6	120.0	28.2	65	206	409
Streams	468,985,764.4	263,836,779.5	523,126,748.2	2,762	3,562,543,890	409
Danceability (%)	67.4	70.0	14.7	23	96	409
Valence (%)	51.2	51.0	23.6	4	97	409
Energy (%)	64.4	66.0	16.1	14	97	409

Figure 1 – Summary Statistics of Key Metrics

As we can see in figure 1, there is a large variation in the values collected for number of streams between all the songs in the dataset. Furthermore, it is worth noting the mean BPM of '122.6', displaying the average beats per minute across all the most popular songs on Spotify. Using this metric along with 'danceability_%' and 'energy_%', I will investigate whether there is a correlation between certain song attributes and higher stream counts.

To explore the correlations between the different metrics within my dataset, I created a colour coded correlation matrix of song attributes. I did this using **'knit', 'corrplot' and 'gt'** packages which enabled me to format a HTML table with custom colour coding, styling and labelling. I then made sure that the figures were rounded to 3 significant figures so that the relations between metrics could be identified easily. This graphic is shown below in figure 2.

Colour Coded Correlation Matrix of Song Attributes

Metric	streams	bpm	danceability_%	energy_%	acousticness_%	valence_%
streams	1	-0.0257	-0.0933	-0.0365	-0.00575	-0.051
bpm	-0.0257	1	-0.149	0.0135	-0.0112	0.0297
danceability_%	-0.0933	-0.149	1	0.158	-0.242	0.391
energy_%	-0.0365	0.0135	0.158	1	-0.553	0.35
acousticness_%	-0.00575	-0.0112	-0.242	-0.553	1	-0.0632
valence_%	-0.051	0.0297	0.391	0.35	-0.0632	1

Figure 2 – Colour Coded Correlation Matrix of Song Attributes

In figure 2 above, we can observe that there is a correlation coefficient of 0.391 between danceability and valence, indicating a moderate positive correlation. This is suggesting that songs that are more danceable tend to have a higher valence (measure of musical positivity). Danceability also has a positive correlation with energy of 0.158 which indicates that more energetic songs tend to be more danceable. Interestingly, there are many negative correlations across the matrix, especially between streams and all the metrics shown above. While they are weak negative correlations, they infer that higher streams do not strongly correlate with danceability, energy, or valence. This could therefore imply that popularity (measured by streams) may not have a direct relationship with these attributes, and that differences among artists could be a better level for explaining variations in song success.

Next, we analyse the trends between BPM, number of streams and danceability level through an interactive scatter plot using **'plotly' and 'ggplot2'**. First the mean values for variables 'bpm' and 'streams' are calculate using mean() function, with 'na.rm = TRUE'. This ensures any missing values are excluded from the calculation. I used geom_point() with a specified transparency (alpha) for better visibility in the graph. The moving average was calculated using the **'zoo'** package and added over the graph in blue. As well as this, a horizontal line for mean BPM is shown in red using geom_hline(). Therefore, as shown in figure 3 below by the moving average line, most successful Spotify songs average between 100-150 BPM. This graphic is interactive, enabling us to hover over each data point to reveal additional information about each song, including the track name, number of streams, BPM, energy level, and danceability.

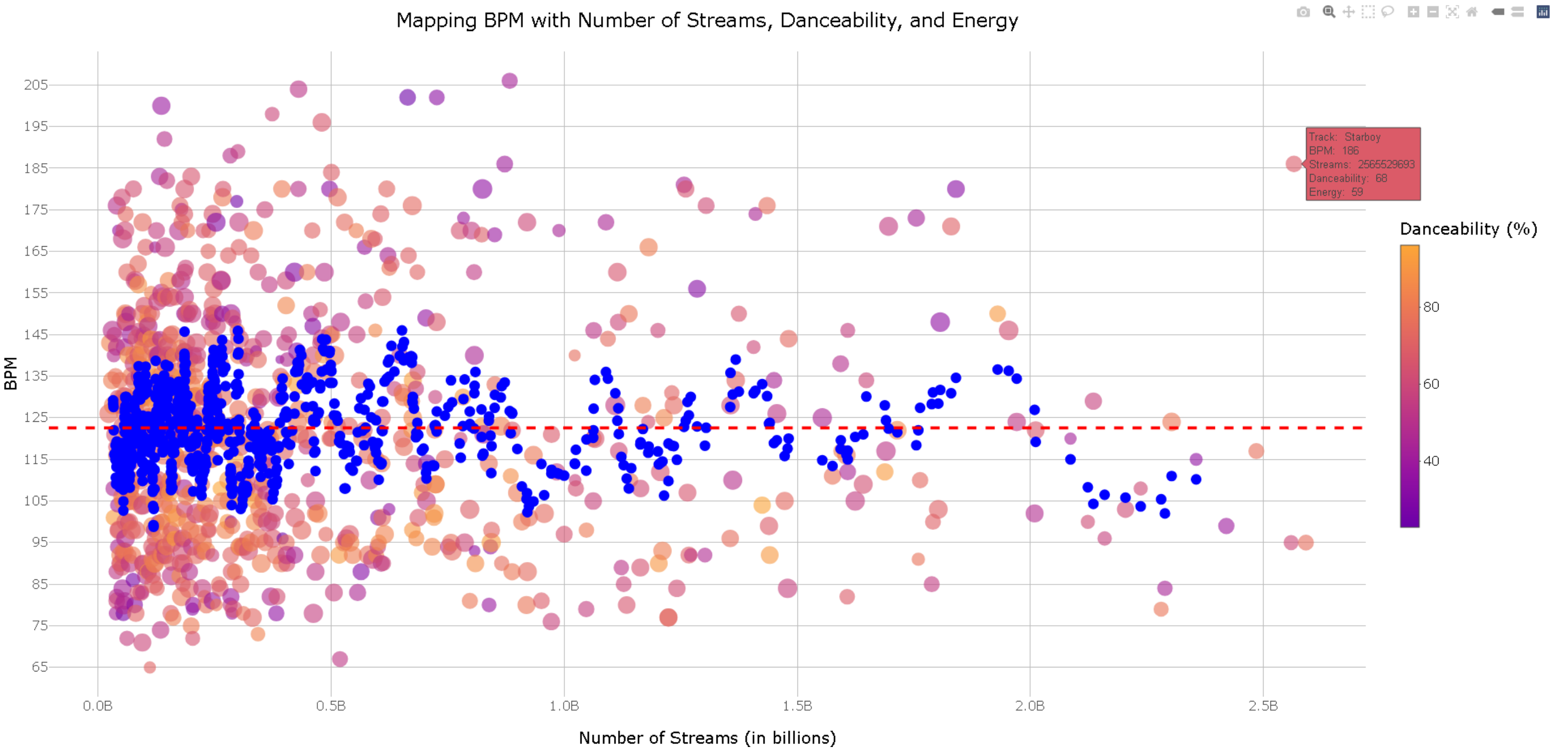


Figure 3 – Interactive scatter plot displaying the relationship between BPM, number of streams and energy

By analysing the colour and size of the points, it can be inferred that tracks with higher energy and danceability levels tend to be more popular with a greater number of streams. It is worth recognising the large array of outliers beyond the moving average BPM line (in blue), suggesting extremely popular tracks with unusual BPM's or varying genres of music that don't align with generic trends in the dataset.

Hierarchical Modelling

Hierarchical modelling known as multilevel modelling, allows us to account for nested structures in the dataset through modelling data at various levels. In our case with this Spotify dataset, streams are likely influenced not only by the specific song characteristics such as BPM and danceability but also by factors varying between artists. Using a Hierarchical Model with artists as a level, we can examine how baseline popularity differs between artists. This approach allows us to compare these findings with a Simple Model that does not account for artist-level variability.

I tailored my Hierarchical Model with 'streams' as the outcome variable, song-level predictors of 'BPM' and 'danceability_%' and artist-level random effects. I used the

'lme4' package with **'lmerTest'** and **'kableExtra'** to produce a summary statistics table for the model. The model can be represented by the equation:

$$Streams_{ij} = \beta_0 + \beta_1(BPM_{ij}) + \beta_2(Danceability\%_{ij}) + u_j + \epsilon_{ij}$$

$Streams_{ij}$ represents the number of streams for song i by artist j. $\beta_0$is the overall intercept which represents the average streams across all songs and artists. $\beta_1$and β_2 are the fixed effects for BPM and danceability, capturing their average effect across songs. u_j refers to the random effect for artist j, representing the deviation of artist j's baseline popularity from the overall average (β_0). ϵ_{ij} displays the residual error, therefore capturing the variation specific to each individual song. This was modelled in R as: streams ~ bpm + danceability_% + (1 | first_artist).

Hierarchical Model Summary

Term	Estimate	Std. Error	t value	P-value
(Intercept)	754800000	127100000	5.9370	3.348e-09
bpm	-551400	635400	-0.8678	0.3856
`danceability_%`	-3344000	1276000	-2.6210	0.008829
sd__(Intercept)	2.24e+08	NA	NA	NA
sd__Observation	468500000	NA	NA	NA

Figure 4 – Hierarchical Model

As we can see from the summary statistics table above, BPM estimate shows a value of -551400 with a p-value of 0.3856. This negative estimate suggests a one-unit increase in BPM slightly reduces stream counts by roughly 551, 400 streams. However, the high p-value shows that this value is not statistically significant. Therefore, BPM does not have a reliable impact on streams in this model. When we look at the coefficient for danceability highlighted in figure 4 above, it suggests that a one-unit increase in danceability percentage reduces streams by 3.34 million on average. The p-value is 0.008829 and is therefore statistically significant, indicating that this effect is meaningful and consistent in the model. This model proves that artists-specific baseline popularity does effect stream counts for the top songs in the dataset.

I then created a simple linear model without the random effects of artist-level variability to compare how well each model fits the data. After fitting both models, I plotted them using **'ggplot2'** package and formatted the scales axes in billions using the **'scales'** package and the label_number() function. The data was combined using both **'broom'** and **'broom.mixed'** which allowed for me to tidy the model outputs. In order to compare these models side by side, I utilised the **'gridExtra'** package to arrange the plots in a grid as shown below in figure 5 using grid.arrange().

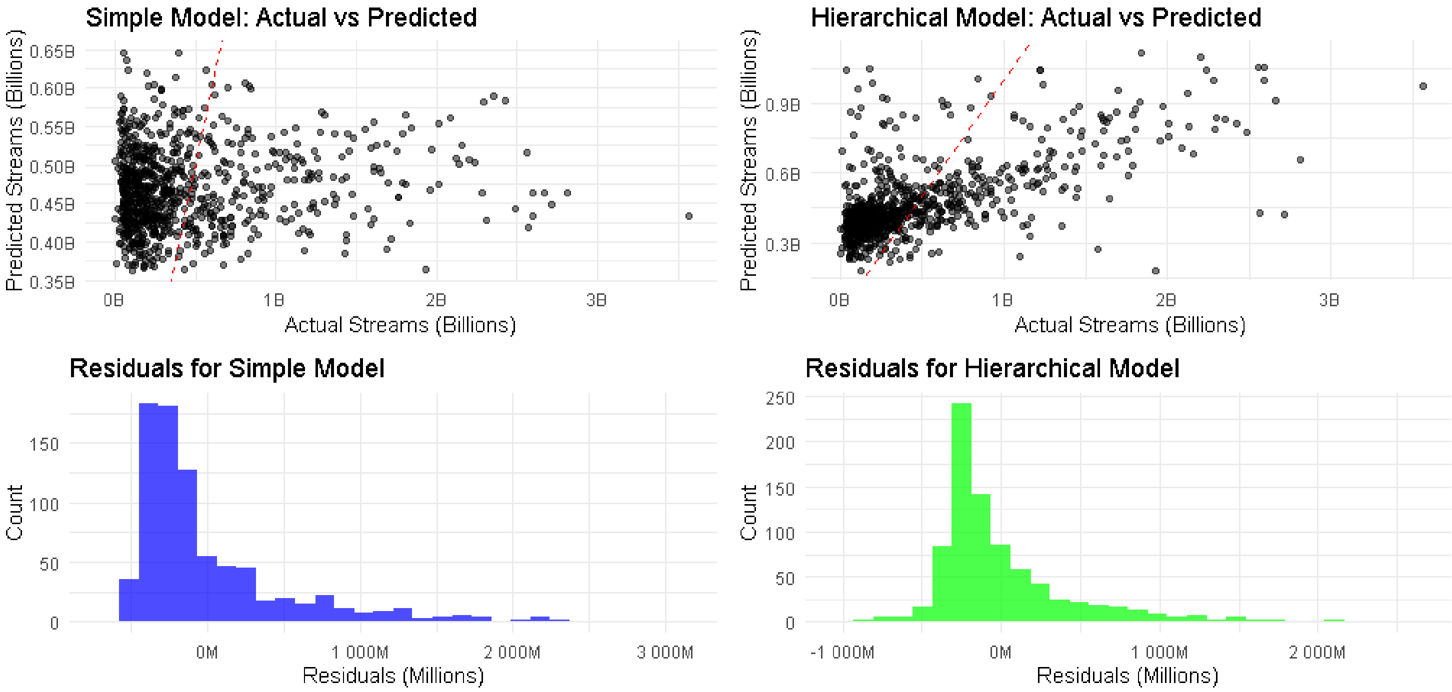


Figure 5 – Simple Model vs Hierarchical Model

The Simple Model shown on the left of figure 5 focuses only on musical atrtributes 'danceability' and 'BPM' to predict number of streams. The scatterplot shows a wide spread of predicted streams against actual streams. The distribution is very random and many predictions are lower than actual streams, suggesting there are possible limitations in capturing the complexity of the data. Futhermore, the histogram (bottom-left) of the residuals shows a right skewed distribution, implying this model underestimates stream counts. The peak around 0 does suggest that some predictions are accurate, but as the data tails off on the right, this indicates a systematic issue of some stream counts being overestimated.

On the other hand, the Hierarchical Model is much more consistent. Through incorporating artist-level variability, the scatter plot in figure 5 displays a tighter alignmenet between predicted and actual streams. The hierarhical model more accutrately demonstrates the trend, reducing the spread of predictions and enhancing the predictive accuracy of the model. The residuals for the Hierarchical Model show a much more symmetric distribution. The histogram is more centrerd around zero with fewer extreme residuals. This model's residuals are more normally distributed than the Simple Model with less skewed data points, indicating this model makes accurate predictions and captures variability more effectively.

Model Comparison Statistics

Model	AIC	BIC	R^2 (Marginal)	R^2 (Conditional)
Simple Model	**35077.20**	**35096.02**	0.0103005	NA
Hierarchical Model	**34944.94**	**34968.46**	0.0089209	0.1932357

Figure 6 – Model Comparison Statistics

Using both the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC), we can compare the performance of the models. I used the **'performance'** package in R to compute the values in the table in figure 6 above. As we can see, the Hierarchical Model has a lower AIC and BIC level indicating that it is a favoured model over the Simple Model, providing a better fit for the data with less complexity. Both R squared values are low, suggesting that neither model explains much of the variance in stream counts according to the predictors. The Hierarchical Model's conditional R squared value of **0.1932357** implies it is capturing more variance when accounting for artist-level variability. Overall, the results support the preference of using a Hierarchical Model to predict streaming counts, as it balances fit and complexity better than the Simple Model.

Limitations

The dataset contains many outliers with some popular songs far exceeding the average in terms of stream counts especially. This could disproportionately affect the model's results. As new data points are added to the dataset, they will need to be standardised against time to give an accurate representation of the song's performance in relation to certain musical attributes over a given time period.

Given the magnitude of the figures used in the analyses, such as streams in the billions, this can make it harder to interpret residual data and other diagnostics in a meaningful way. Scaling and standardising the data can help to improve the model's numerical stability and improve convergence in iterative models such as the hierarchical model used, ensuring unbiased results.

The lack of a genre column in the dataset means that the overall analysis fails to examine trends related to different music genres. Including genre as an additional level in the hierarchical model would be beneficial to understanding whether certain genres influence streams, BPM or danceability. Combining a genre data set with this existing dataset and matching the genre types to the relevant song names would allow for a richer analysis and understanding of which types of songs perform better.

Conclusion

The results of my exploratory data analysis revealed that a hierarchical model using multi-level variability is an effective approach for understanding the streaming success of songs on Spotify. Certain musical attributes like danceability showed statistically significant impact on streams, but artist-specific influences proved to be a crucial factor in accurately modelling stream counts.

The hierarchical model's inclusion of artist-level variability provided a more consistent and accurate predictive model. Graphically, it displayed a tighter alignment between predicted and actual stream counts, with a more centred residual pattern.

Concluding our hypothesis, the findings support the **alternative hypothesis:** the inclusion of artist-level variability does improve the model's ability to explain the relationship between musical attributes and stream counts. This is shown by the hierarchical model's lower AIC and BIC values, which demonstrate a better fit in comparison to the simple model.

Through using this analysis and incorporating additional levels such as genre, region or country, this model could provide insights into which attributes drive popularity in specific genres and how this varies across regions. This addition could be useful to music producers or streaming platforms that want to target certain regions with genre-specific songs. The model could understand which areas appreciate certain attributes like energy or danceability and create targeted marketing campaigns off the insights generated. Furthermore, if the hierarchical model is improved with a time-based level, it could display trends in musical attributes over time. Analysts could track seasonal patterns and understand which song attributes maintain popularity over a given period.

Overall, the hierarchical model provides a powerful foundation for studying key, complex, nested structures in the Spotify dataset. This approach to data analysis can help stakeholders make informed data-driven business decisions in the music industry.

Appendices

Appendix A: Spotify Stock Price

Google Finance - Spotify Technology SA (SPOT) Stock Price & News. (2024). Google Finance. https://www.google.com/finance/quote/SPOT:NYSE?sa=X&ved=2ahUKEwj5iYuW066JAxXMSUEAHU5uLScQ3ecFegQIWBAg

Appendix B: Spotify Dataset

Abdullah, M. (2024). *Spotify Most Streamed Songs*. Kaggle.com. https://www.kaggle.com/datasets/abdulszz/spotify-most-streamed-songs

Spotify Most Streamed CSV file – from Kaggle.com.

Appendix C : R code

```
library(readr)
library(ggplot2)
library(plotly)
library(readr)
library(dplyr)
library(tibble)
Spotify_2024_data <- read_csv("C:/Users/joshh/Downloads/Spotify Most Streamed Songs.csv")
View(Spotify_2024_data)
## note to change the column names so they are more presentable
str(Spotify_2024_data)
Spotify_2024_data <- na.omit(Spotify_2024_data) # Removes rows with NA values
Spotify_2024_data$`danceability_%` <- as.numeric(Spotify_2024_data$`danceability_%`)

##------------------------ Data Wrangling -----------------------------##
unique(Spotify_2024_data$streams)
Spotify_2024_data$streams <- as.numeric(gsub(",", "", as.character(Spotify_2024_data$streams)))
sum(is.na(Spotify_2024_data$streams))
Spotify_2024_data <- na.omit(Spotify_2024_data)
Spotify_2024_data$streams <- as.numeric(gsub(",", "", Spotify_2024_data$streams))
```

```
# Extract the first artist (artists separated by commas)
Spotify_2024_data <- Spotify_2024_data %>%
  mutate(first_artist = sapply(strsplit(`artist(s)_name`, ","), `[`, 1))

# Calculate the number of unique first artists
num_unique_first_artists <- Spotify_2024_data %>%
 summarise(unique_first_artists = n_distinct(first_artist)) %>%
  pull(unique_first_artists)
num_unique_first_artists

library(zoo)
Spotify_2024_data <- Spotify_2024_data %>%
  arrange(streams) %>%
  mutate(moving_avg_bpm = zoo::rollmean(bpm, k = 10, fill = NA))

Spotify_2024_data$'energy_%' <- as.numeric(Spotify_2024_data$'energy_%')

##--------------------------- Making Tables ----------------------------##
library(dplyr)
library(knitr)
library(kableExtra)

# Create individual summaries for each relevant metric
bpm_summary <- data.frame(
  Metric = "BPM",
  Mean = mean(Spotify_2024_data$bpm, na.rm = TRUE),
  Median = median(Spotify_2024_data$bpm, na.rm = TRUE),
  SD = sd(Spotify_2024_data$bpm, na.rm = TRUE),
  Min = min(Spotify_2024_data$bpm, na.rm = TRUE),
  Max = max(Spotify_2024_data$bpm, na.rm = TRUE),
  Unique_Artists = n_distinct(Spotify_2024_data$first_artist)
)

streams_summary <- data.frame(
  Metric = "Streams",
  Mean = mean(Spotify_2024_data$streams, na.rm = TRUE),
  Median = median(Spotify_2024_data$streams, na.rm = TRUE),
```

```
SD = sd(Spotify_2024_data$streams, na.rm = TRUE),
 Min = min(Spotify_2024_data$streams, na.rm = TRUE),
 Max = max(Spotify_2024_data$streams, na.rm = TRUE),
 Unique_Artists = n_distinct(Spotify_2024_data$first_artist)
)

danceability_summary <- data.frame(
 Metric = "Danceability (%)",
 Mean = mean(Spotify_2024_data$`danceability_%`, na.rm = TRUE),
 Median = median(Spotify_2024_data$`danceability_%`, na.rm = TRUE),
 SD = sd(Spotify_2024_data$`danceability_%`, na.rm = TRUE),
 Min = min(Spotify_2024_data$`danceability_%`, na.rm = TRUE),
 Max = max(Spotify_2024_data$`danceability_%`, na.rm = TRUE),
 Unique_Artists = n_distinct(Spotify_2024_data$first_artist)
)

valence_summary <- data.frame(
 Metric = "Valence (%)",
 Mean = mean(Spotify_2024_data$`valence_%`, na.rm = TRUE),
 Median = median(Spotify_2024_data$`valence_%`, na.rm = TRUE),
 SD = sd(Spotify_2024_data$`valence_%`, na.rm = TRUE),
 Min = min(Spotify_2024_data$`valence_%`, na.rm = TRUE),
 Max = max(Spotify_2024_data$`valence_%`, na.rm = TRUE),
 Unique_Artists = n_distinct(Spotify_2024_data$first_artist)
)

energy_summary <- data.frame(
 Metric = "Energy (%)",
 Mean = mean(Spotify_2024_data$`energy_%`, na.rm = TRUE),
 Median = median(Spotify_2024_data$`energy_%`, na.rm = TRUE),
 SD = sd(Spotify_2024_data$`energy_%`, na.rm = TRUE),
 Min = min(Spotify_2024_data$`energy_%`, na.rm = TRUE),
 Max = max(Spotify_2024_data$`energy_%`, na.rm = TRUE),
 Unique_Artists = n_distinct(Spotify_2024_data$first_artist)
)

# Combine the individual summaries into a single table
```

```
summary_table <- bind_rows(bpm_summary, streams_summary, danceability_summary, valence_summary,
energy_summary)

# Format the table to display large numbers with commas
summary_table$Mean <- scales::comma(summary_table$Mean)
summary_table$Median <- scales::comma(summary_table$Median)
summary_table$SD <- scales::comma(summary_table$SD)
summary_table$Min <- scales::comma(summary_table$Min)
summary_table$Max <- scales::comma(summary_table$Max)

# Present the table using kableExtra for clean formatting
summary_table %>%
 kbl(col.names = c("Metric", "Mean", "Median", "Standard Deviation", "Min", "Max", "Unique Artists"),
   caption = "Summary Statistics for Key Metrics") %>%
 kable_styling(full_width = F, bootstrap_options = c("striped", "hover", "condensed", "responsive"))

# Making Colour Coded Correlation Matrix of Song Attributes
library(gt)
library(corrplot)

correlation_table <- correlation_df %>%
 rownames_to_column(var = "Metric") %>%
 mutate(across(-Metric, ~ cell_spec(signif(.x, 3),  # Round to 3 significant figures
                   color = "white",
                   background = spec_color(.x, option = "D", end = 0.9)))) %>%
 knitr::kable(format = "html", escape = FALSE,
       caption = "Colour Coded Correlation Matrix of Song Attributes") %>%
 kable_styling(full_width = FALSE, bootstrap_options = c("striped", "hover", "condensed")) %>%
 row_spec(0, bold = TRUE) %>%
 column_spec(1, bold = TRUE)

# Display the table in an HTML viewer
correlation_table

# Create a correlation plot
corrplot(correlation_matrix, method = "color", type = "upper",
     tl.col = "black", tl.srt = 45, addCoef.col = "black")
```

View(correlation_matrix)

##-------------------- Making Scatterplot graph -------------------------##

library(ggplot2)

 # Create a boxplot to show the distribution of streams by key

```
ggplot(Spotify_2024_data, aes(x = factor(key), y = streams)) +
 geom_boxplot(aes(fill = factor(key))) +  # Boxplot with key on x-axis and streams on y-axis
 labs(title = "Song Streams by Key", x = "Key of the Song", y = "Number of Streams") +
 scale_y_continuous(labels = scales::comma) +  # Format y-axis with commas
 theme_minimal() +
 theme(legend.position = "none")  # Remove legend as it's redundant
```

```
mean_streams_by_key <- Spotify_2024_data %>%
 group_by(key) %>%
 summarize(mean_streams = mean(streams, na.rm = TRUE))
```

 # Create a bar plot to show mean streams by key

```
ggplot(mean_streams_by_key, aes(x = factor(key), y = mean_streams)) +
 geom_bar(stat = "identity", aes(fill = factor(key))) +  # Bar plot
 labs(title = "Mean Song Streams by Key", x = "Key of the Song", y = "Mean Number of Streams") +
 scale_y_continuous(labels = scales::comma) +  # Format y-axis with commas
 theme_minimal() +
 theme(legend.position = "none")
```

 # Create the scatter plot with streams on the x-axis

```
ggplot(Spotify_2024_data, aes(x = streams, y = bpm, color = `danceability_%`)) +
 geom_point(alpha = 0.7, size = 3) +
 scale_color_gradient(low = "yellow", high = "blue") + # Color gradient for danceability
 labs(title = "Mapping BPM with Number of Streams and Danceability",
    x = "Number of Streams",
    y = "BPM",
    color = "Danceability (%)") +
 theme_minimal() +
 theme(axis.text.x = element_text(size = 10),
     axis.text.y = element_text(size = 10))
```

```
## Iteration

ggplot(Spotify_2024_data, aes(x = streams, y = bpm, color = `danceability_%`)) +
 geom_point(alpha = 0.7, size = 3) +
 scale_color_gradient(low = "yellow", high = "blue") +
 labs(title = "Mapping Songs BPM with their Number of Streams and Danceability",
    x = "Number of Streams",
    y = "BPM",
    color = "Danceability (%)") +
 theme_minimal() +
 theme(axis.text.x = element_text(size = 10),
     axis.text.y = element_text(size = 10))

##------------- Iteration for interactive scatter plot ---------------------##

Spotify_2024_data$streams <- as.numeric(gsub(",", "", as.character(Spotify_2024_data$streams)))
# Remove rows with NA values in streams
Spotify_2024_data <- na.omit(Spotify_2024_data)

install.packages("scales")
library(scales)



####### test without log scale ##########

Spotify_2024_data$`energy_%` <- as.numeric(as.character(Spotify_2024_data$`energy_%`))

# Calculate averages for BPM and streams
bpm_avg <- mean(Spotify_2024_data$bpm, na.rm = TRUE)
streams_avg <- mean(Spotify_2024_data$streams, na.rm = TRUE)

# Create the plot without log scale
p <- ggplot(Spotify_2024_data, aes(x = streams, y = bpm, color = `danceability_%`,
                  text = paste("Track: ", track_name,
                        "<br>BPM: ", bpm,
                        "<br>Streams: ", streams,
                        "<br>Danceability: ", `danceability_%`))) +
```

```
  geom_point(alpha = 0.7, size = 3) +
  geom_smooth(method = "loess", se = FALSE, color = "black", linetype = "dashed") +  # Trend line
  geom_hline(yintercept = bpm_avg, linetype = "dotted", color = "red") +  # Horizontal average line for BPM
  geom_vline(xintercept = streams_avg, linetype = "dotted", color = "red") +  # Vertical average line for streams
  scale_color_viridis_c(option = "plasma") +  # Color palette
  labs(title = "Mapping BPM with Number of Streams and Danceability",
     x = "Number of Streams",
     y = "BPM",
     color = "Danceability (%)") +
  theme_minimal(base_size = 15) +
  theme(legend.position = "right", plot.title = element_text(hjust = 0.5))

# Replace the log scale with a continuous scale for x-axis
p <- p +
  scale_x_continuous(labels = scales::label_number(scale_cut = scales::cut_short_scale()),
             limits = c(0, max(Spotify_2024_data$streams) * 1.05),
             expand = expansion(mult = c(0.05, 0.05))) +  # Adjusting the limits and adding padding
  scale_size_continuous(range = c(3, 8))  # Standardize dot sizes

# Convert to interactive plotly object
interactive_plot <- ggplotly(p, tooltip = "text")

# Save and display the interactive plot
saveWidget(interactive_plot, "interactive_scatter_plot_no_log.html", selfcontained = TRUE)
browseURL("interactive_scatter_plot_no_log.html")

### Plot with trends ###
library(zoo)
Spotify_2024_data <- Spotify_2024_data %>%
  arrange(streams) %>%
  mutate(moving_avg_bpm = zoo::rollmean(bpm, k = 10, fill = NA))
Spotify_2024_data$'energy_%' <- as.numeric(Spotify_2024_data$'energy_%')

# Create a fresh plot using the updated data with the moving_avg_bpm column
p <- ggplot(Spotify_2024_data, aes(x = streams, y = bpm, color = `danceability_%`, size = `energy_%`,
              text = paste("Track: ", track_name,
                    "<br>BPM: ", bpm,
```

```
                                   "<br>Streams: ", streams,
                                   "<br>Danceability: ", `danceability_%`,
                                   "<br>Energy: ", `energy_%`))) +
  geom_point(alpha = 0.7) +  # Plot the points
  scale_color_viridis_c(option = "plasma") +  # Color scale
  scale_size_continuous(range = c(2, 6)) +  # Size scale for dots

  # Updated label formatting for the x-axis
  scale_x_continuous(labels = scales::label_number(scale_cut = scales::cut_si(""))) +
   labs(title = "Mapping BPM with Number of Streams, Danceability, and Energy",
     x = "Number of Streams",
     y = "BPM",
     color = "Danceability (%)",
     size = "Energy (%)") +
  theme_minimal(base_size = 15) +
  theme(legend.position = "bottom", plot.title = element_text(hjust = 0.5)) +

  # Add the moving average line
  geom_line(aes(x = streams, y = moving_avg_bpm), color = "red", size = 1.2)  # Add moving average line

# Convert to interactive plotly object
interactive_plot <- ggplotly(p, tooltip = "text")

# Save the updated interactive plot
saveWidget(interactive_plot, "interactive_scatter_plot_with_moving_average.html", selfcontained = TRUE)
browseURL("interactive_scatter_plot_with_moving_average.html")

## The blue trend scatter can demonstrate that most successful spotify songs average between 100-150 BPM.
##

# Calculate mean values for BPM and Streams
mean_bpm <- mean(Spotify_2024_data$bpm, na.rm = TRUE)
mean_streams <- mean(Spotify_2024_data$streams, na.rm = TRUE)

# Assuming Spotify_2024_data has been prepared with moving_avg_bpm
p <- ggplot(Spotify_2024_data, aes(x = streams, y = bpm,
                      color = `danceability_%`,
```

```
                    size = `energy_%`,
                    text = paste("Track: ", track_name,
                            "<br>BPM: ", bpm,
                            "<br>Streams: ", streams,
                            "<br>Danceability: ", `danceability_%`,
                            "<br>Energy: ", `energy_%`))) +
  geom_point(alpha = 0.6) +  # Slightly larger point size and increased transparency
  scale_color_viridis_c(option = "plasma", begin = 0.2, end = 0.8) +  # Color scale with adjusted range
  scale_size(range = c(2, 6), guide = "none") +  # Size scale without guide

  # Custom formatting for the x-axis to show billions
  scale_x_continuous(labels = label_number(scale = 1e-9, suffix = "B"),
              breaks = seq(0, max(Spotify_2024_data$streams), by = 0.5e9)) +  # Custom breaks
  scale_y_continuous(breaks = seq(min(Spotify_2024_data$bpm), max(Spotify_2024_data$bpm), by = 10)) +  #
Custom breaks for BPM

  labs(title = "Mapping BPM with Number of Streams, Danceability, and Energy",
     x = "Number of Streams (in billions)",
     y = "BPM",
     color = "Danceability (%)",
     size = "Energy (%)") +
  theme_minimal(base_size = 15) +
  theme(legend.position = "bottom",
      plot.title = element_text(hjust = 0.5),
      panel.grid.major = element_line(color = "grey", size = 0.5),  # Add grid lines
      panel.grid.minor = element_blank()) +

  # Add the moving average line with improved visibility
  geom_line(aes(x = streams, y = moving_avg_bpm), color = "blue", size = 2) +  # Thicker line for moving average
  geom_point(aes(x = streams, y = moving_avg_bpm), color = "blue", size = 3, shape = 21, fill = "blue") +  # Points for the
moving average

  geom_hline(yintercept = mean_bpm, color = "red", linetype = "dashed", size = 1) +  # Mean BPM line
   # Optional linear trend line
  geom_smooth(method = "lm", se = FALSE, color = "orange", linetype = "solid", size = 1.5)

# Convert to interactive plotly object
interactive_plot <- ggplotly(p, tooltip = "text")
```

```
# Save the updated interactive plot
saveWidget(interactive_plot, "interactive_scatter_plot_with_moving_average.html", selfcontained = TRUE)
browseURL("interactive_scatter_plot_with_moving_average.html")

##------------------------------- Hierarchial Modelling -------------------------------##

library(lme4)
# Plot random effects for artists
ranef_df <- as.data.frame(ranef(hierarchical_model)$first_artist)
colnames(ranef_df) <- c("Intercept")
ranef_df$Artist <- rownames(ranef(hierarchical_model)$first_artist)

# Plot Random Effects (Artist Intercepts)
library(ggplot2)
ggplot(ranef_df, aes(x = Artist, y = Intercept)) +
  geom_point() +
  coord_flip() +  # Flipping for readability
  labs(title = "Random Intercepts by Artist", x = "Artist", y = "Intercept (BPM)") +
  theme_minimal()

# Plot the relationship between log(streams) and bpm
ggplot(Spotify_2024_data, aes(x = log_streams, y = bpm)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", color = "red", se = FALSE) +
  labs(title = "BPM vs Log(Streams)", x = "Log(Streams)", y = "BPM") +
  theme_minimal()

# Plot model fit: predicted vs actual BPM
Spotify_2024_data$predicted_bpm <- predict(hierarchical_model)
ggplot(Spotify_2024_data, aes(x = bpm, y = predicted_bpm)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red") +  # Line y=x for perfect fit
  labs(title = "Actual vs Predicted BPM", x = "Actual BPM", y = "Predicted BPM") +
  theme_minimal()
```

```
# Log-transforming the streams variable
Spotify_2024_data$log_streams <- log(Spotify_2024_data$streams + 1)  # adding 1 to avoid log(0)


# Rebuild hierarchical model with log-streams
hierarchical_model <- lmer(bpm ~ log_streams + `danceability_%` + (1 | first_artist), data = Spotify_2024_data)


# Display the summary
summary(hierarchical_model)
hierarchical_model_streams <- lmer(streams ~ bpm + `danceability_%` + (1 | first_artist), data = Spotify_2024_data)
summary(hierarchical_model_streams)


# Load the lmerTest package
library(lmerTest)
# Refit the model using lmerTest
hierarchical_model_with_p <- lmer(streams ~ bpm + `danceability_%` + (1 | first_artist),
                   data = Spotify_2024_data)


# Summary will now include p-values
summary(hierarchical_model_with_p)


library(broom.mixed)
library(kableExtra)


# Tidy the hierarchical model output
tidy_model <- tidy(hierarchical_model_streams)


tidy_model_table <- tidy_model %>%
 select(term, estimate, std.error, statistic, p.value) %>%
 rename(
  Term = term,
  Estimate = estimate,
  `Std. Error` = std.error,
  `t value` = statistic,
  `P-value` = p.value
 ) %>%
 # Apply significant figure formatting to 4 digits
 mutate(across(c(Estimate, `Std. Error`, `t value`, `P-value`), ~ signif(.x, 4)))
```

```r
# Create a formatted table with kableExtra
tidy_model_table %>%
  knitr::kable("html", escape = FALSE, caption = "Hierarchical Model Summary") %>%
  kable_styling(full_width = FALSE, bootstrap_options = c("striped", "hover", "condensed")) %>%
  column_spec(1, bold = TRUE) %>%
  row_spec(0, bold = TRUE, color = "white", background = "#8DA0CB") %>%  # Style header
  add_header_above(c(" " = 1, "Model Summary Statistics" = 4))

# Tidy the hierarchical model output
tidy_hierarchical <- tidy(hierarchical_model_streams)


tidy_hierarchical_table <- tidy_hierarchical %>%
  select(term, estimate, std.error, statistic, p.value) %>%
  rename(
    Term = term,
    Estimate = estimate,
    `Std. Error` = std.error,
    `t value` = statistic,
    `P-value` = p.value
  )%>%
mutate(across(c(Estimate, `Std. Error`, `t value`, `P-value`), ~ signif(.x, 4)))



# Highlight p-value and estimate for danceability
tidy_hierarchical_table <- tidy_hierarchical_table %>%
  mutate(
    Estimate = ifelse(Term == "`danceability_%`", cell_spec(Estimate, color = "blue", bold = TRUE), Estimate),
    `P-value` = ifelse(Term == "`danceability_%`", cell_spec(`P-value`, color = "red", bold = TRUE), `P-value`)
  )

# Create a formatted table with kableExtra
tidy_hierarchical_table %>%
  knitr::kable("html", escape = FALSE, caption = "Hierarchical Model Summary") %>%
  kable_styling(full_width = FALSE, bootstrap_options = c("striped", "hover", "condensed")) %>%
  column_spec(1, bold = TRUE) %>%
  row_spec(0, bold = TRUE, color = "white", background = "#6BAED6")  # Header styling
```

- fitting more models for hierarchical modelling with stream -

 # Remove NA values from Spotify_2024_data and then fit the model

Spotify_2024_data <- na.omit(Spotify_2024_data)

hierarchical_model_streams <- lmer(streams ~ bpm + `danceability_%` + (1 | first_artist), data = Spotify_2024_data)

 # Ensure predictions are made only on rows used in the model

Spotify_2024_data <- Spotify_2024_data[!is.na(Spotify_2024_data$bpm) &
!is.na(Spotify_2024_data$`danceability_%`),]

 # Assign predicted values

Spotify_2024_data$predicted_streams <- predict(hierarchical_model_streams, newdata = Spotify_2024_data)

library(ggplot2)

 # Plot actual vs predicted streams with formatted axes in billions

```
ggplot(Spotify_2024_data, aes(x = streams, y = predicted_streams)) +
 geom_point(alpha = 0.5) +  # Scatter plot of actual vs predicted
 geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +  # Line y=x for perfect fit
 scale_x_continuous(
  labels = label_number(scale = 1e-9, suffix = "B"),  # Convert to billions with "B" suffix
  breaks = pretty_breaks(n = 5)  # Set approximately 5 breaks on the x-axis
 ) +
 scale_y_continuous(
  labels = label_number(scale = 1e-9, suffix = "B"),  # Convert to billions with "B" suffix
  breaks = pretty_breaks(n = 5)  # Set approximately 5 breaks on the y-axis
 ) +
 labs(
  title = "Actual vs Predicted Streams",
  x = "Actual Streams (Billions)",
  y = "Predicted Streams (Billions)"
 ) +
 theme_minimal(base_size = 15) +
 theme(
  axis.title.x = element_text(margin = margin(t = 10)),  # Adds padding to the x-axis title
  axis.title.y = element_text(margin = margin(r = 10))   # Adds padding to the y-axis title
```

```r
 )

## Fitting a simple linear model and comparing it against the hierarchical model ##
# Simple linear model without random effects
simple_model_streams <- lm(streams ~ bpm + `danceability_%`, data = Spotify_2024_data)

# Hierarchical model with random intercepts for artists
hierarchical_model_streams <- lmer(streams ~ bpm + `danceability_%` + (1 | first_artist), data = Spotify_2024_data)

# Compare AIC and BIC values for both models
aic_values <- c(AIC(simple_model_streams), AIC(hierarchical_model_streams))
bic_values <- c(BIC(simple_model_streams), BIC(hierarchical_model_streams))

library(performance)

# Calculate R^2 values for the hierarchical model
r2_hierarchical <- r2_nakagawa(hierarchical_model_streams)

# Calculate R^2 for the simple linear model
r2_simple <- summary(simple_model_streams)$r.squared

# Create a summary table for AIC, BIC, and R^2 values
comparison_table <- data.frame(
  Model = c("Simple Model", "Hierarchical Model"),
  AIC = aic_values,
  BIC = bic_values,
  `R^2 (Marginal)` = c(r2_simple, r2_hierarchical$R2_marginal),
  `R^2 (Conditional)` = c(NA, r2_hierarchical$R2_conditional)
)

colnames(comparison_table) <- c("Model", "AIC", "BIC", "R^2 (Marginal)", "R^2 (Conditional)")

# Format the table with kableExtra
library(knitr)
library(kableExtra)
library(gridExtra)
library(scales)
```

```
comparison_table %>%
  knitr::kable("html", escape = FALSE, caption = "Model Comparison for Streams") %>%
  kable_styling(full_width = FALSE, bootstrap_options = c("striped", "hover", "condensed")) %>%
  row_spec(0, bold = TRUE, color = "white", background = "#6BAED6") %>%
  add_header_above(c("Model Comparison Statistics" = 6)) %>%  # Change to 6 to match the columns
  column_spec(3:4, bold = TRUE, width = "8em") %>%  # Optional width adjustment
  row_spec(1:2, color = "black")  # Adjust row color for readability


## Both models predicted vs actual side by side ##
# Remove rows with missing values in relevant columns
Spotify_2024_data_clean <- Spotify_2024_data %>%
  filter(!is.na(streams) & !is.na(bpm) & !is.na(`danceability_%`))

# Fit models with the cleaned data
simple_model_streams <- lm(streams ~ bpm + `danceability_%`, data = Spotify_2024_data_clean)
hierarchical_model_streams <- lmer(streams ~ bpm + `danceability_%` + (1 | first_artist), data =
Spotify_2024_data_clean)

# Make predictions with the cleaned dataset
Spotify_2024_data_clean$predicted_simple <- predict(simple_model_streams)
Spotify_2024_data_clean$predicted_hierarchical <- predict(hierarchical_model_streams)

# Ensure predictions are made only for rows used in the model
Spotify_2024_data$predicted_simple <- predict(simple_model_streams, newdata = Spotify_2024_data)
Spotify_2024_data$predicted_hierarchical <- predict(hierarchical_model_streams, newdata = Spotify_2024_data)

# Simple Model Plot with Billions on Axes
plot_simple <- ggplot(Spotify_2024_data, aes(x = streams / 1e9, y = predicted_simple / 1e9)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +
  scale_x_continuous(labels = label_number(scale = 1, suffix = "B")) +
  scale_y_continuous(labels = label_number(scale = 1, suffix = "B")) +
  labs(title = "Simple Model: Actual vs Predicted", x = "Actual Streams (Billions)", y = "Predicted Streams (Billions)") +
  theme_minimal()
```

 # Hierarchical Model Plot with Billions on Axes

plot_hierarchical <- ggplot(Spotify_2024_data, aes(x = streams / 1e9, y = predicted_hierarchical / 1e9)) +

geom_point(alpha = 0.5) +

geom_abline(slope = 1, intercept = 0, color = "red", linetype = "dashed") +

scale_x_continuous(labels = label_number(scale = 1, suffix = "B")) +

scale_y_continuous(labels = label_number(scale = 1, suffix = "B")) +

labs(title = "Hierarchical Model: Actual vs Predicted", x = "Actual Streams (Billions)", y = "Predicted Streams (Billions)") +

theme_minimal()

 # Calculate residuals for both models

Spotify_2024_data$residuals_simple <- Spotify_2024_data$streams - Spotify_2024_data$predicted_simple

Spotify_2024_data$residuals_hierarchical <- Spotify_2024_data$streams - Spotify_2024_data$predicted_hierarchical

 # Plot residuals for Simple Model with improved x-axis readability

residuals_simple_plot <- ggplot(Spotify_2024_data, aes(x = residuals_simple / 1e6)) +

geom_histogram(bins = 30, fill = "blue", alpha = 0.7) +

scale_x_continuous(labels = label_number(scale = 1, suffix = "M"), breaks = pretty_breaks(n = 5)) +

labs(title = "Residuals for Simple Model", x = "Residuals (Millions)", y = "Count") +

theme_minimal()

 # Plot residuals for Hierarchical Model with improved x-axis readability

residuals_hierarchical_plot <- ggplot(Spotify_2024_data, aes(x = residuals_hierarchical / 1e6)) +

geom_histogram(bins = 30, fill = "green", alpha = 0.7) +

scale_x_continuous(labels = label_number(scale = 1, suffix = "M"), breaks = pretty_breaks(n = 5)) +

labs(title = "Residuals for Hierarchical Model", x = "Residuals (Millions)", y = "Count") +

theme_minimal()

 # Arrange all plots in a grid of 2x2

grid.arrange(plot_simple, plot_hierarchical, residuals_simple_plot, residuals_hierarchical_plot, ncol = 2)

 # Coefficient Comparison

simple_model_tidy <- tidy(simple_model_streams)

hierarchical_model_tidy <- tidy(hierarchical_model_streams)

 # Add model labels

simple_model_tidy$model <- "Simple Model" hierarchical_model_tidy$model <- "Hierarchical Model"

 # Combine data

combined_tidy <- rbind(simple_model_tidy, hierarchical_model_tidy)

 # Plot effect sizes side by side

ggplot(combined_tidy, aes(x = term, y = estimate, color = model)) +

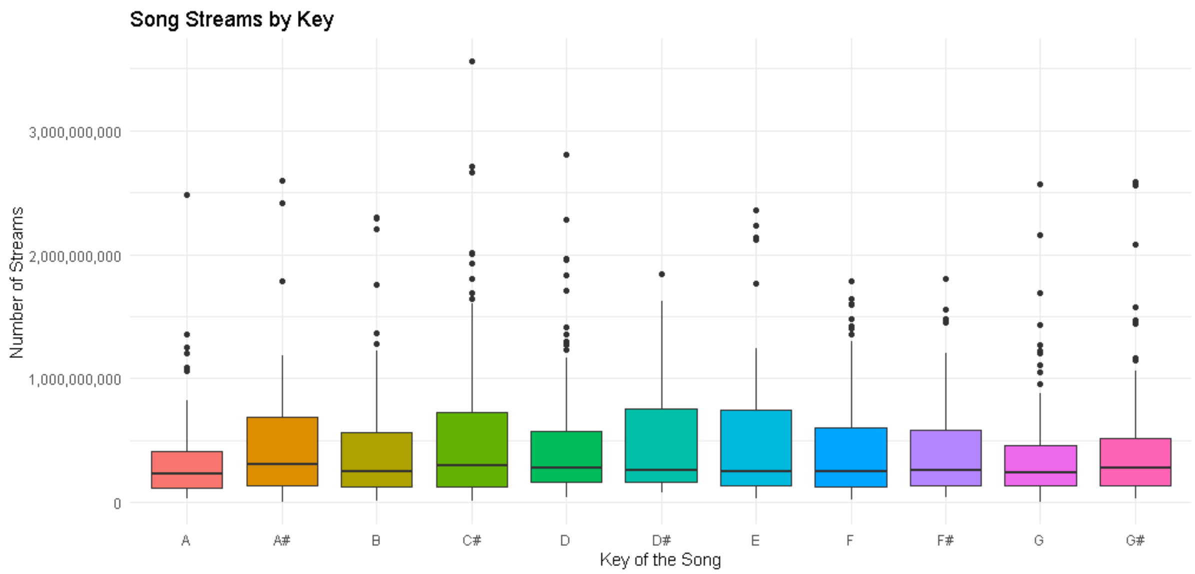
geom_point(position = position_dodge(width = 0.5)) +

geom_errorbar(aes(ymin = estimate - std.error, ymax = estimate + std.error), width = 0.2, position = position_dodge(width = 0.5)) +

labs(title = "Coefficient Comparison: Simple vs Hierarchical Model", x = "Predictors", y = "Estimate") +

theme_minimal()

Appendix D : Box Plot displaying song streams by key



This graphic can be used to explain how popular songs in the Spotify data set are grouped by 'Key' of song.

Appendix E: ChatGPT prompts

Model Comparison Statistics

AIC	BIC	R^2 (Marginal)	R^2 (Conditional)
35077.20	35096.02	0.0103005	NA
34944.94	34968.46	0.0089209	0.1932357

these values compare both of my models - how can i interpret them

The table you've provided summarizes key statistical metrics for both the Simple Model and the Hierarchical Model. Here's how to interpret each statistic:

Model Comparison Statistics Interpretation

1. AIC (Akaike Information Criterion):
   - Simple Model AIC: 35077.20
   - Hierarchical Model AIC: 34944.94
   - Interpretation: The AIC is used to compare models, with lower values indicating a better fit when considering model complexity. The Hierarchical Model has a lower AIC value, suggesting it fits the data better than the Simple Model, taking into account the number of parameters used.
2. BIC (Bayesian Information Criterion):
   - Simple Model BIC: 35096.02
   - Hierarchical Model BIC: 34968.46
   - Interpretation: Similar to AIC, the BIC penalizes model complexity but does so more harshly. A lower BIC value also indicates a better model. Again, the Hierarchical Model is favored here as it has a lower BIC, suggesting it is the more parsimonious model for the data.
3. R^2 (Marginal):
   - Simple Model R^2: 0.0103005
   - Hierarchical Model R^2: 0.0089209
   - Interpretation: The R^2 value indicates the proportion of variance in the dependent variable that can be explained by the model. Both models have low R^2 values, indicating that neither model explains much of the variance in stream counts based on the predictors. However, the Simple Model has a slightly higher marginal R^2, suggesting it explains marginally more variance than the Hierarchical Model.
4. R^2 (Conditional):
   - Simple Model R^2: NA
   - Hierarchical Model R^2: 0.1932357
   - Interpretation: The conditional R^2 reflects the proportion of variance explained by the entire model, including both fixed and random effects in the Hierarchical Model. The Hierarchical Model has a conditional R^2 of approximately 0.193, indicating that it explains about 19.3% of the variance in the data, which is a substantial improvement compared to the marginal R^2 values of both models.

rewrite this for me: This could therefore imply that popularity (measured by streams) might not directly relate to these attributes and artist variability could explain popularity in a better way

This suggests that popularity, as indicated by streams, may not have a direct correlation with these attributes, and that differences among artists could better account for variations in popularity.

what can we infer from this correlation matrix?

The correlation matrix you've provided illustrates the relationships between various song attributes, including streams, beats per minute (bpm), danceability, energy, acousticness, and valence. Here's a summary of what we can infer from the matrix:

1. Strong Positive Correlation:
   - Danceability and Valence: The correlation coefficient of 0.391 indicates a moderate positive correlation. This suggests that songs that are more danceable tend to have a higher valence (a measure of musical positiveness).
   - Energy: Danceability also has a positive correlation with energy (0.158), indicating that more energetic songs tend to be more danceable.
2. Weak Positive Correlation:
   - BPM and Energy: There is a slight positive correlation (0.135) between bpm and energy, suggesting that faster songs may have slightly higher energy levels.
   - BPM and Danceability: A weak negative correlation (-0.149) indicates that as the bpm increases, the danceability may slightly decrease, although this relationship is not strong.
3. Negative Correlations:
   - Streams with Danceability, Energy, Acousticness, and Valence: All these correlations are negative but weak (ranging from -0.051 to -0.093). This indicates that higher streams do not strongly correlate with danceability, energy, or valence, implying that popularity (as measured by streams) might not directly relate to these attributes.
   - Acousticness and Energy: A stronger negative correlation (-0.553) indicates that as acousticness increases, energy tends to decrease, which is common since more acoustic songs may be softer or slower.