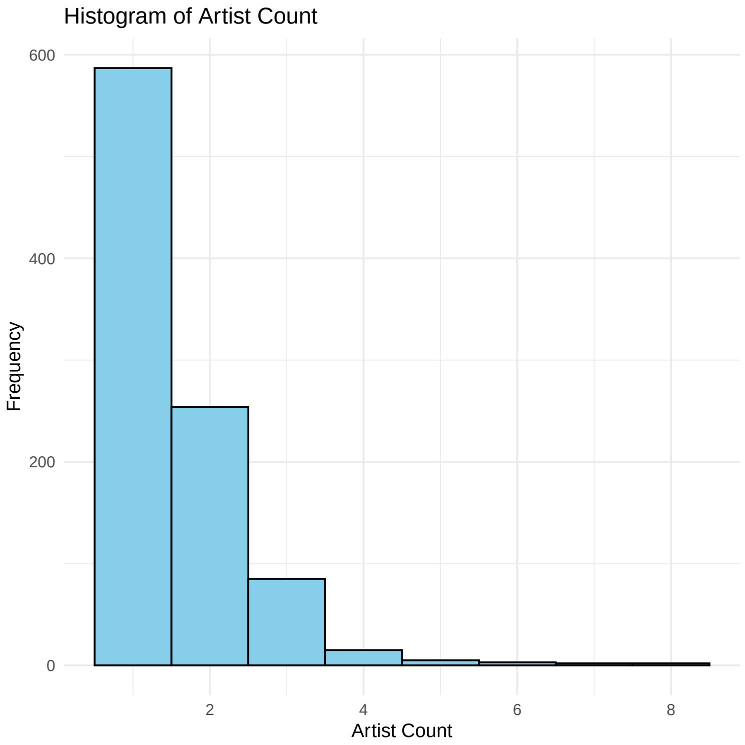
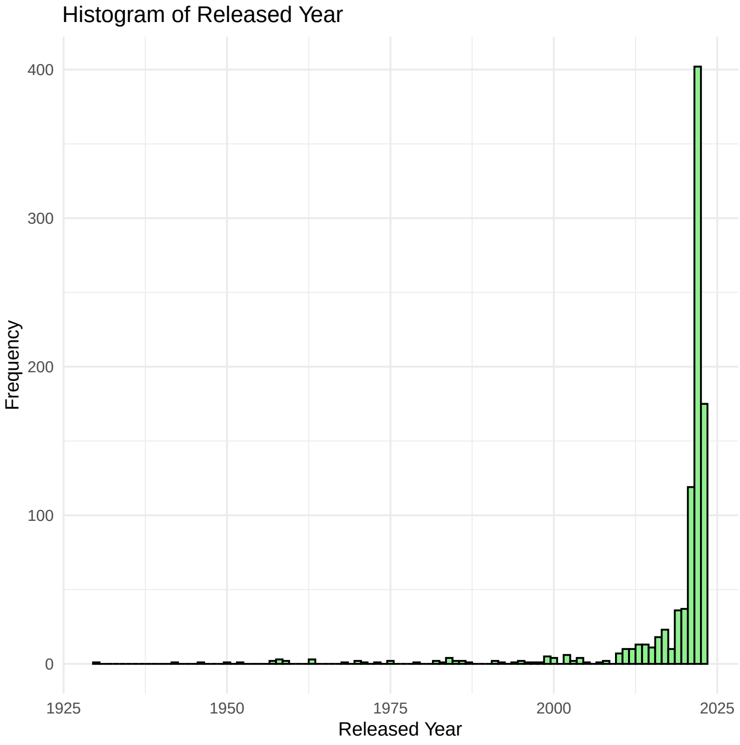
March 1, 2024

The results below are generated from an R script.

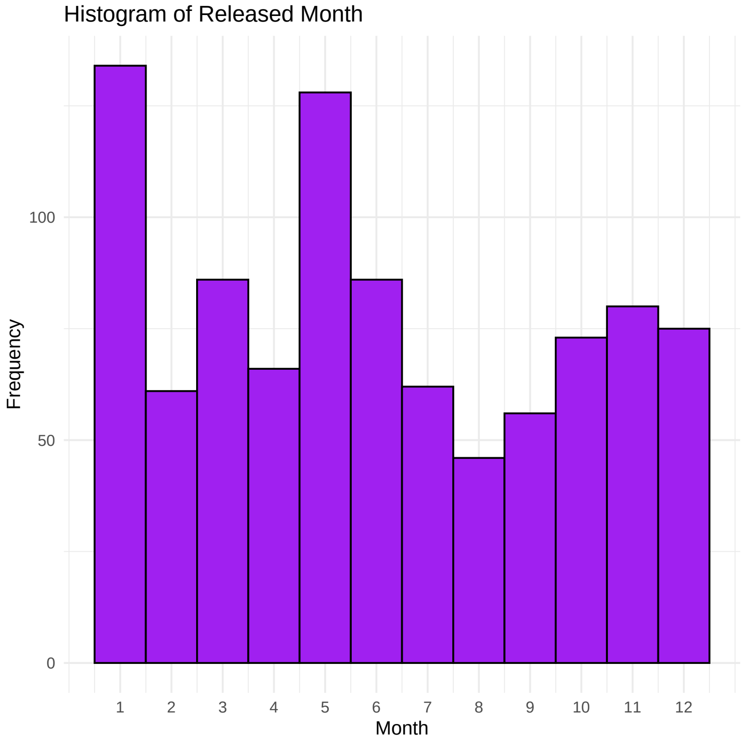
```
# Load required library
library(ggplot2)
library(dplyr)
library(MASS)


# load the specific diagram
spotify_data <- read.csv("spotify-2023.csv")

# creates histogram for artist_count
ggplot(spotify_data, aes(x = artist_count)) +
  geom_histogram(binwidth = 1, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Artist Count",
       x = "Artist Count",
       y = "Frequency") +
  theme_minimal()
```



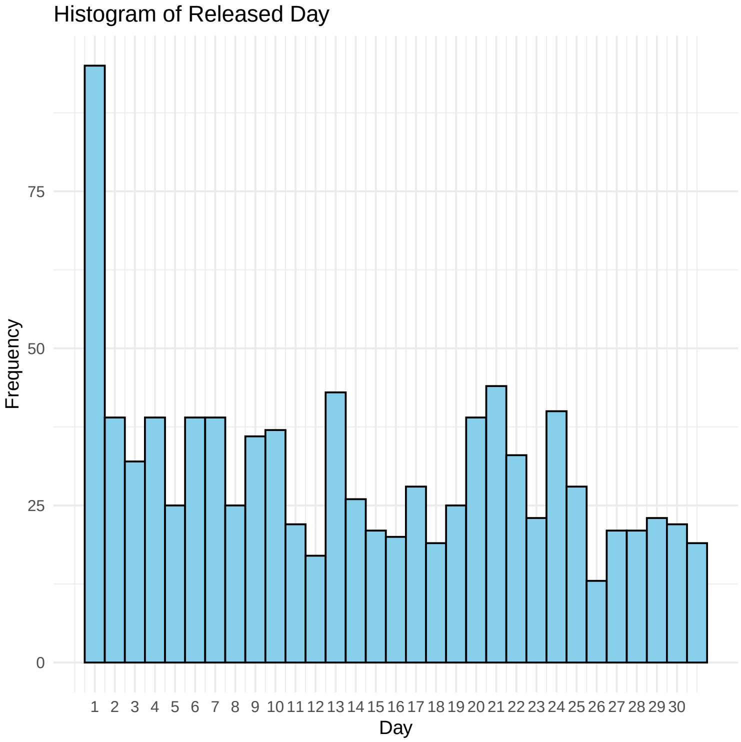
```r
# creates histogram for released_year
ggplot(spotify_data, aes(x = released_year)) +
  geom_histogram(binwidth = 1, fill = "lightgreen", color = "black") +
  labs(title = "Histogram of Released Year",
       x = "Released Year",
       y = "Frequency") +
  theme_minimal()
```



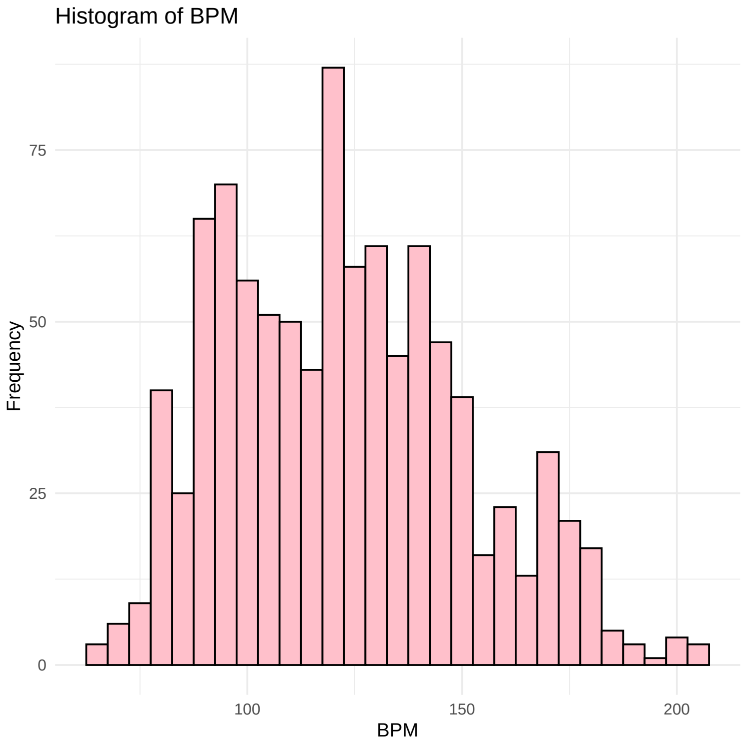
```r
# creats histogram for released_month
ggplot(spotify_data, aes(x = released_month)) +
  geom_histogram(binwidth = 1, fill = "purple", color = "black") +
  labs(title = "Histogram of Released Month",
       x = "Month",
       y = "Frequency") +
  scale_x_continuous(breaks = 1:12) +  # Set breaks to show each month
  theme_minimal()
```



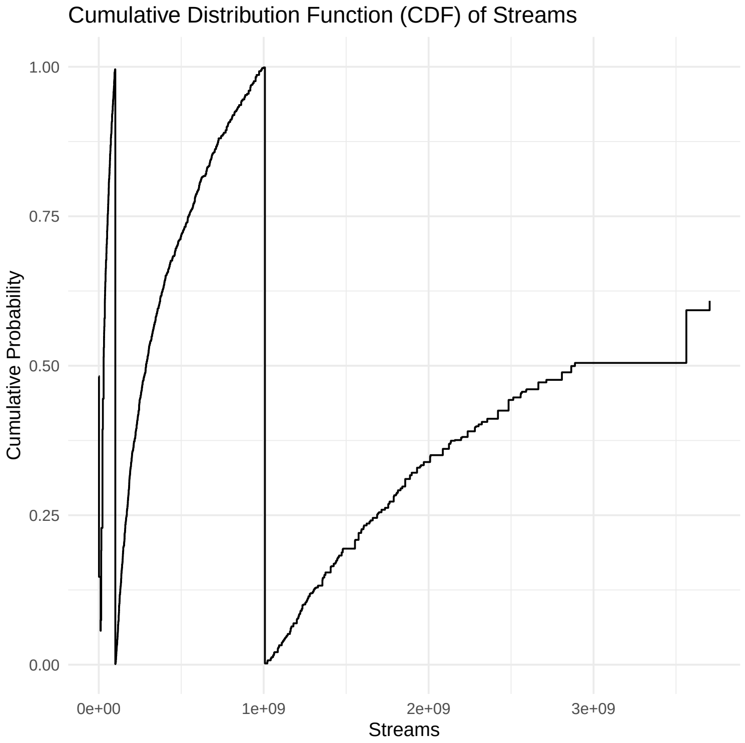
```r
# creates histogram for released_day
ggplot(spotify_data, aes(x = released_day)) +
  geom_histogram(binwidth = 1, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Released Day",
       x = "Day",
       y = "Frequency") +
  scale_x_continuous(breaks = 1:30) +  # Set breaks to show each day
  theme_minimal()
```



```
# creates histogram for bpm
ggplot(spotify_data, aes(x = bpm)) +
  geom_histogram(binwidth = 5, fill = "pink", color = "black") +
  labs(title = "Histogram of BPM",
       x = "BPM",
       y = "Frequency") +
  theme_minimal()
```



```
# Calculate statistics for each variable
statistics <- summarise(spotify_data,
                        artist_count_mean = mean(artist_count),
                        released_year_mean = mean(released_year),
                        released_month_mean = mean(released_month),
                        released_day_mean = mean(released_day),
                        in_spotify_charts_mean = mean(in_spotify_charts),
                        bpm_mean = mean(bpm),
                        artist_count_mode = as.numeric(names(sort(table(artist_count), decreasing = TRU
                        released_year_mode = as.numeric(names(sort(table(released_year), decreasing = T
                        released_month_mode = as.numeric(names(sort(table(released_month), decreasing =
                        released_day_mode = as.numeric(names(sort(table(released_day), decreasing = TRU
                        in_spotify_charts_mode = as.numeric(names(sort(table(in_spotify_charts), decreas
                        bpm_mode = as.numeric(names(sort(table(bpm), decreasing = TRUE)[1])),
                        bpm_sd = sd(bpm),
                        streams_quantile_25 = quantile(streams, 0.25),
                        streams_quantile_75 = quantile(streams, 0.75),
                        bpm_quantile_25 = quantile(bpm, 0.25),
                        bpm_quantile_75 = quantile(bpm, 0.75),
)

## Error in ‘summarise()‘:
## i In argument: ‘streams_quantile_25 = quantile(streams, 0.25)‘.
## Caused by error in ‘(1 - h) * qs[i]‘:
## ! non-numeric argument to binary operator

# print statistics
print(statistics)

##   artist_count_mean released_year_mean released_month_mean released_day_mean
## 1          1.556723           2018.289            6.038866          13.94433
##   in_spotify_charts_mean bpm_mean artist_count_mode released_year_mode
## 1               12.02206 122.5536                 1               2022
##   released_month_mode released_day_mode in_spotify_charts_mode bpm_mode  bpm_sd
## 1                   1                 1                      0      120 28.0696
##   streams_quantile_25 streams_quantile_75 bpm_quantile_25 bpm_quantile_75
## 1           141636175           673869022           99.75          140.25

# PMF

# compare two scenarios in your data using a PMF
total_songs <- nrow(spotify_data)

# songs released in 2023
songs_2023 <- spotify_data %>%
  filter(released_year == 2023) %>%
  nrow()

pmf_2023 <- songs_2023 / total_songs

# songs released in other years
songs_other_years <- spotify_data %>%
  filter(released_year != 2023) %>%
  nrow()

pmf_other_years <- songs_other_years / total_songs

# display PMFs
print(paste("Probability of a song being released in 2023:", pmf_2023))

## [1] "Probability of a song being released in 2023: 0.183630640083945"

print(paste("Probability of a song being released in other years:", pmf_other_years))

## [1] "Probability of a song being released in other years: 0.816369359916055"

# CDF

# sort the data by streams variable
streams <- spotify_data[order(spotify_data$streams), ]

# calculate the cumulative probabilities
streams$cdf <- seq(1, nrow(streams)) / nrow(streams)

# convert streams variable to numeric, removes scientific notation
streams$streams <- as.numeric(streams$streams)

## Warning:  NAs introduced by coercion

#  logarithmic scaling streams variable
streams$log_streams <- log(streams$streams)

# plot the CDF
ggplot(streams, aes(x = streams, y = cdf)) +
  geom_step() +
  labs(title = "Cumulative Distribution Function (CDF) of Streams",
       x = "Streams",
       y = "Cumulative Probability") +
  theme_minimal()

## Warning:  Removed 1 row containing missing values (‘geom_step()‘).
```

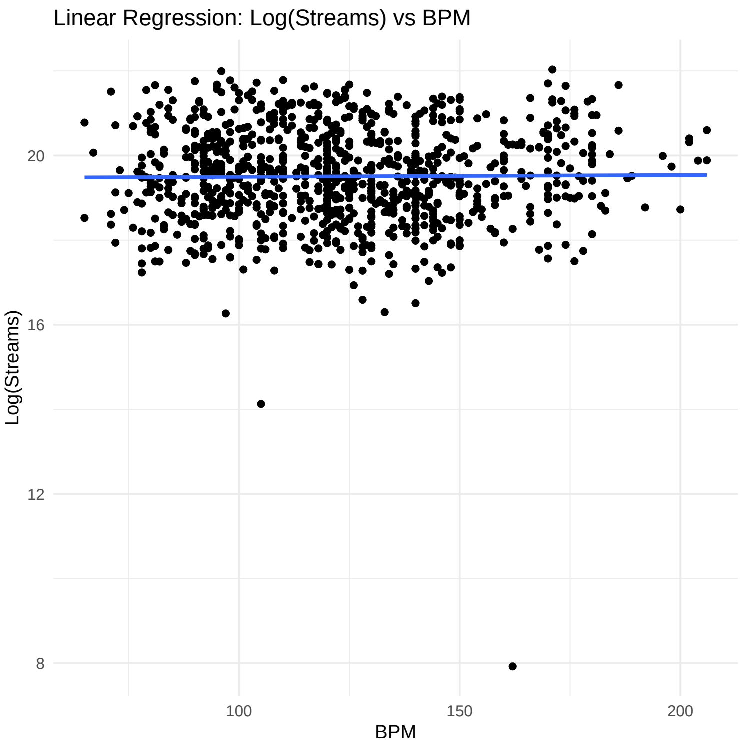


```
# Analytical Distribution

# remove rows with missing values
spotify_data$streams <- as.numeric(as.character(spotify_data$streams))

## Warning:  NAs introduced by coercion

spotify_data$bpm <- as.numeric(as.character(spotify_data$bpm))

# removes invalid values from streams and bpm
spotify_data <- spotify_data[complete.cases(spotify_data$streams, spotify_data$bpm), ]

# apply logarithmic transformation to streams
spotify_data$log_streams <- log(spotify_data$streams)

# perform linear regression
regression_model <- lm(log_streams ~ bpm, data = spotify_data)

# summary of regression model
summary(regression_model)

##
## Call:
```

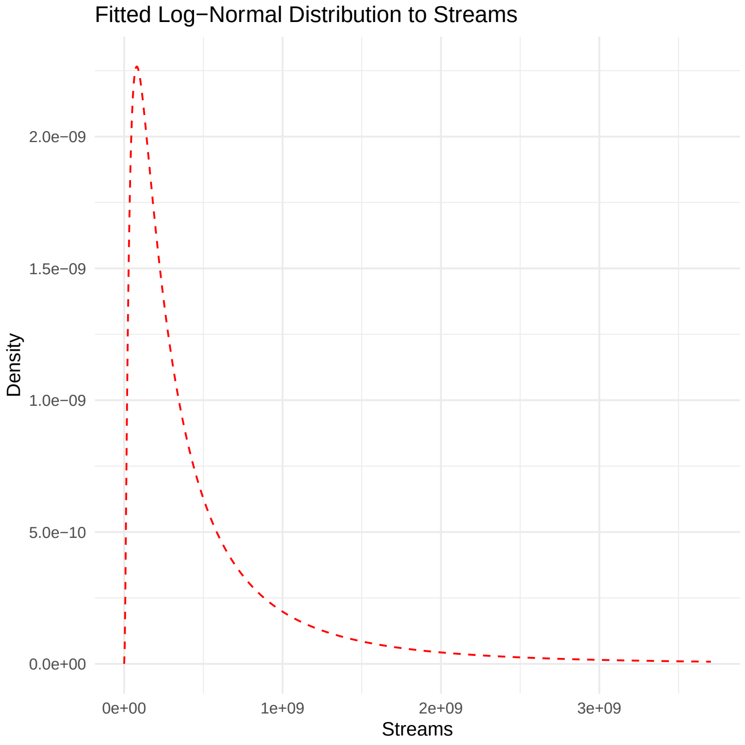
```
## lm(formula = log_streams ~ bpm, data = spotify_data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5995  -0.7411  -0.0192   0.8185   2.5057
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.946e+01  1.666e-01 116.801   <2e-16 ***
## bpm         4.189e-04  1.325e-03   0.316    0.752
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 1.147 on 950 degrees of freedom
## Multiple R-squared:  0.0001052, Adjusted R-squared:  -0.0009473
## F-statistic: 0.09996 on 1 and 950 DF,  p-value: 0.7519
```

```
# plot the regression line
ggplot(spotify_data, aes(x = bpm, y = log_streams)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Linear Regression: Log(Streams) vs BPM",
       x = "BPM",
       y = "Log(Streams)") +
  theme_minimal()
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```

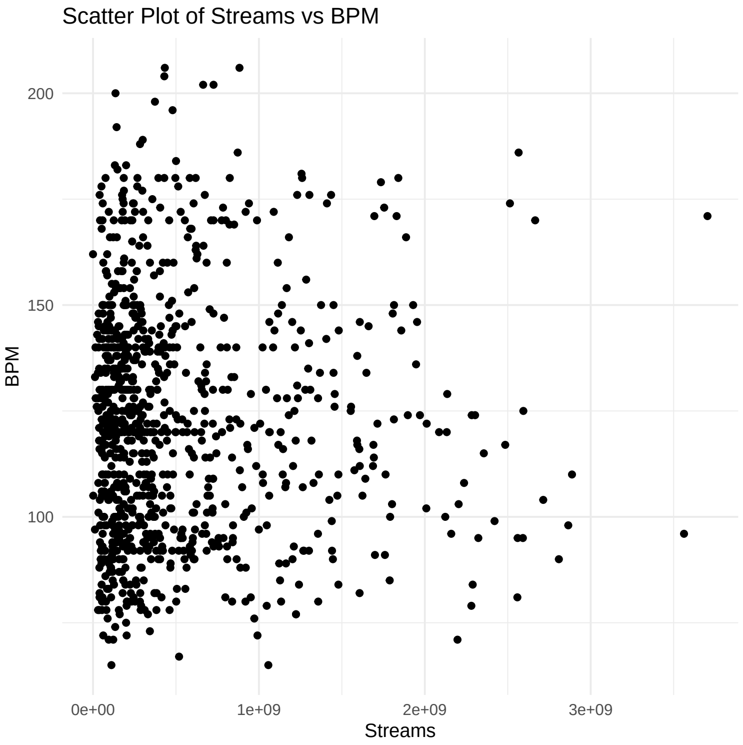


```r
# plot the fitted log-normal distribution
ggplot() +
  geom_line(data = data.frame(x = x, y = y), aes(x = x, y = y), color = "red", linetype = "dashed") +
  labs(title = "Fitted Log-Normal Distribution to Streams",
       x = "Streams",
       y = "Density") +
  theme_minimal()
```

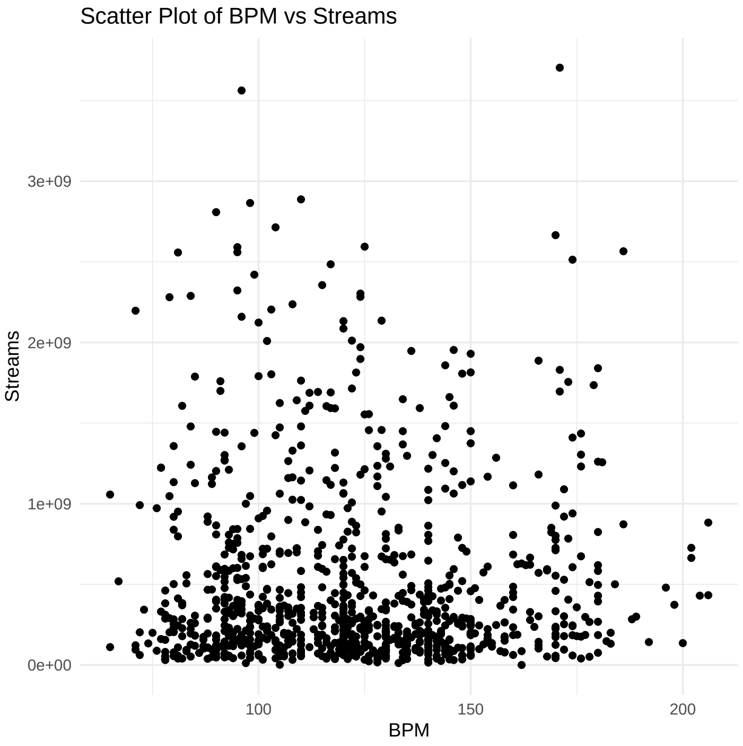


```r
# scatter plot

# convert streams and bpm variables to numeric
spotify_data$streams <- as.numeric(as.character(spotify_data$streams))
spotify_data$bpm <- as.numeric(as.character(spotify_data$bpm))

#  scatter plot for streams vs bpm
scatter_plot_streams_bpm <- ggplot(spotify_data, aes(x = streams, y = bpm)) +
  geom_point() +
  labs(title = "Scatter Plot of Streams vs BPM",
       x = "Streams",
       y = "BPM") +
  theme_minimal()

#  scatter plot for bpm vs streams
scatter_plot_bpm_streams <- ggplot(spotify_data, aes(x = bpm, y = streams)) +
  geom_point() +
  labs(title = "Scatter Plot of BPM vs Streams",
       x = "BPM",
       y = "Streams") +
  theme_minimal()
```

```r
# Print both scatter plots
print(scatter_plot_streams_bpm)
```

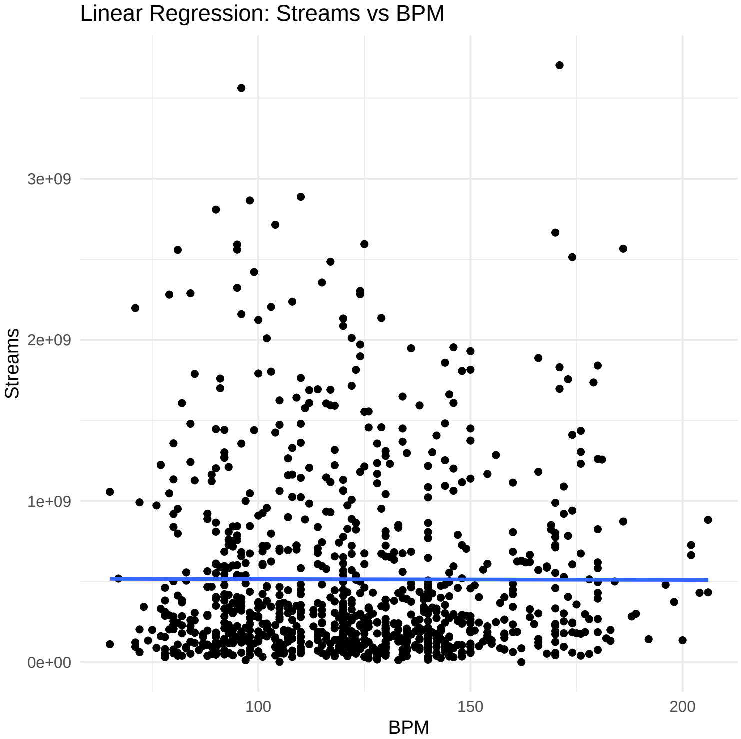


```r
print(scatter_plot_bpm_streams)
```



```
# conducted test

# perform Pearson’s correlation test
cor_results <- cor.test(spotify_data$streams, spotify_data$bpm)

# print the test result
cor_results

##
##  Pearson’s product-moment correlation
##
## data:  spotify_data$streams and spotify_data$bpm
## t = -0.075142, df = 950, p-value = 0.9401
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.06596511  0.06110897
## sample estimates:
##          cor
## -0.002437908

# conduct a regression analysis

# linear regression
regression_model <- lm(streams ~ bpm, data = spotify_data)

# summary of regression model
summary(regression_model)

##
## Call:
## lm(formula = streams ~ bpm, data = spotify_data)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -513636452 -372808926 -223552907  159316140 3192142803
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 520171081   82374227   6.315 4.15e-10 ***
## bpm            -49233     655200  -0.075     0.94
## ---
## Signif. codes:  0 ’***’ 0.001 ’**’ 0.01 ’*’ 0.05 ’.’ 0.1 ’ ’ 1
##
## Residual standard error: 567200000 on 950 degrees of freedom
## Multiple R-squared:  5.943e-06, Adjusted R-squared:  -0.001047
## F-statistic: 0.005646 on 1 and 950 DF,  p-value: 0.9401

# plot the regression line
ggplot(spotify_data, aes(x = bpm, y = streams)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  labs(title = "Linear Regression: Streams vs BPM",
       x = "BPM",
       y = "Streams") +
  theme_minimal()
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



The R session information (including the OS info, R version and all packages used):

```
sessionInfo()

## R version 4.3.2 (2023-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19045)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_United States.utf8  LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8 LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: America/New_York
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] MASS_7.3-60   dplyr_1.1.4   ggplot2_3.4.4
##
## loaded via a namespace (and not attached):
##  [1] Matrix_1.6-1.1   gtable_0.3.4     compiler_4.3.2   highr_0.10       tidyselect_1.2.0
##  [6] tinytex_0.49     splines_4.3.2    scales_1.3.0     yaml_2.3.7       fastmap_1.1.1
## [11] lattice_0.21-9   R6_2.5.1         labeling_0.4.3   generics_0.1.3   knitr_1.45
## [16] tibble_3.2.1     munsell_0.5.0    pillar_1.9.0     rlang_1.1.2      utf8_1.2.4
## [21] xfun_0.41        cli_3.6.1        withr_2.5.2      magrittr_2.0.3   mgcv_1.9-0
## [26] digest_0.6.33    grid_4.3.2       lifecycle_1.0.4  nlme_3.1-163     vctrs_0.6.5
## [31] evaluate_0.23    glue_1.6.2       farver_2.1.1     fansi_1.0.6      colorspace_2.1-0
## [36] rmarkdown_2.25   tools_4.3.2      pkgconfig_2.0.3  htmltools_0.5.7
```

```
Sys.time()
```

```
## [1] "2024-03-01 13:25:49 EST"
```