Text-Mining and Exploratory-Analysis

Harini

01/01/2021

INTRODUCTION

Lyric analysis is slowly finding its way into data science communities as the possibility of predicting "Hit Songs" approaches reality.The project work on text mining techniques on a set of lyrics using the tidy text framework. Tidy datasets have a specific structure in which each variable is a column, each observation is a row, and each type of observational unit is a table. Here, all inferences are purely observational; hence, correlation does not imply causation. I did some minor cleaning and saved the result to a csv file.

DATA INGESTION

Using names() function to see the columns in the data frame

```
names(prince_orignal)
```

```
##  [1] "X"            "text"         "artist"       "song"         "year"
##  [6] "album"        "Release.Date" "US.Pop"       "US.R.B"       "CA"
## [11] "UK"           "IR"           "NL"           "DE"           "AT"
## [16] "FR"           "JP"           "AU"           "NZ"           "peak"
```

X is just a row number and text is the actual lyrics. The other needed fields include song, year, and peakUS.Pop and US.R.B are peak chart positions for the US (Pop and R&B charts), so keep those around as well, and drop all the other fields for now. Set text to lyrics and rename the US columns to the tidyverse style using "_" instead of "." and then store the results in prince.dplyr provides a function called glimpse() that makes it easy to look at the data in a transposed view.dim() function, you see that there are 7 columns and 824 observations. Each observation is a song.

```
prince <- prince_orignal %>%
  select(lyrics = text, song, year, album, peak,
         us_pop = US.Pop, us_rnb = US.R.B)

glimpse(prince[139,])
```

```
## Rows: 1
## Columns: 7
## $ lyrics <chr> "I just can't believe all the things people say, controversy...
## $ song   <chr> "controversy"
## $ year   <int> 1981
## $ album  <chr> "Controversy"
## $ peak   <int> 3
## $ us_pop <chr> "70"
## $ us_rnb <chr> "3"
```

```
dim(prince)
```

```
## [1] 824   7
```

Looking at the lyrics column for one of the songs, you can see how they are structured.

```
str(prince[139, ]$lyrics, nchar.max = 300)
```

```
##  chr "I just can't believe all the things people say, controversy\nAm I Black or White? Am I straight or
gay? Controversy\nDo I believe in God? Do I believe in me? Controversy\nControversy, controversy\nI can't un
derstand human curiosity, controversy\nWas it good for you? Was I what you w"| __truncated__
```

DATA CLEANING

Using gsub() and apply() functions to condition the data.Convert the data frame to a Corpus and Document Term Matrix using the tm text mining package and then use the tm_map() function to do the cleaning.

```r
# function to expand contractions in an English-language source
fix.contractions <- function(doc) {
  # "won't" is a special case as it does not expand to "wo not"
  doc <- gsub("won't", "will not", doc)
  doc <- gsub("can't", "can not", doc)
  doc <- gsub("n't", " not", doc)
  doc <- gsub("'ll", " will", doc)
  doc <- gsub("'re", " are", doc)
  doc <- gsub("'ve", " have", doc)
  doc <- gsub("'m", " am", doc)
  doc <- gsub("'d", " would", doc)
  # 's could be 'is' or could be possessive: it has no expansion
  doc <- gsub("'s", "", doc)
  return(doc)
}

# fix (expand) contractions
prince$lyrics <- sapply(prince$lyrics, fix.contractions)
```

gsub() function and a simple regular expression used to remove special characters.

```r
# function to remove special characters
removeSpecialChars <- function(x) gsub("[^a-zA-Z0-9 ]", " ", x)
# remove special characters
prince$lyrics <- sapply(prince$lyrics, removeSpecialChars)
```

Convert everything to lowercase with the handy tolower() function.

```r
# convert everything to lower case
prince$lyrics <- sapply(prince$lyrics, tolower)
```

Examining the lyrics now shows a nice, cleaner version of the original, raw text.

```r
str(prince[139, ]$lyrics, nchar.max = 300)
```

```
##  chr "i just can not believe all the things people say  controversy am i black or white  am i straight or
gay  controversy do i believe in god  do i believe in me  controversy controversy  controversy i can not und
erstand human curiosity  controversy was it good for you  was i what you wa"| __truncated__
```

Another common step in conditioning data for text mining is called stemming, or breaking down words to their root meaning.For now, take a look at the summary of the prince data frame.

```r
#get facts about the full dataset
summary(prince)
```

```
##     lyrics               song                 year          album          
##  Length:824         Length:824         Min.   :1978   Length:824        
##  Class :character   Class :character   1st Qu.:1989   Class :character  
##  Mode  :character   Mode  :character   Median :1996   Mode  :character  
##                                        Mean   :1995                     
##                                        3rd Qu.:1999                     
##                                        Max.   :2015                     
##                                        NA's   :495                      
##       peak          us_pop             us_rnb         
##  Min.   : 0.00   Length:824         Length:824        
##  1st Qu.: 2.00   Class :character   Class :character  
##  Median : 7.00   Mode  :character   Mode  :character  
##  Mean   :15.48                                        
##  3rd Qu.:19.00                                        
##  Max.   :88.00                                        
##  NA's   :751
```

Here are 37 years of songs, and the lowest charted song (that exists in the dataset) is at position 88. You can also see that there are quite a few NAs for year and peak.Keep the full dataset around in the prince data frame and just filter when needed. Target questions is to look for song trends across time, and the dataset contains individual release years.

```
#create the decade column
prince <- prince %>%
  mutate(decade =
           ifelse(prince$year %in% 1978:1979, "1970s",
           ifelse(prince$year %in% 1980:1989, "1980s",
           ifelse(prince$year %in% 1990:1999, "1990s",
           ifelse(prince$year %in% 2000:2009, "2000s",
           ifelse(prince$year %in% 2010:2015, "2010s",
                  "NA"))))))
```

chart_level attribute, which represents whether a song peaked in the Top 10, Top 100, or did not chart (that is, uncharted). These are mutually exclusive, so Top 100 does not include Top 10 songs.

```
#create the chart level column
prince <- prince %>%
  mutate(chart_level =
           ifelse(prince$peak %in% 1:10, "Top 10",
           ifelse(prince$peak %in% 11:100, "Top 100", "Uncharted")))
```

Create a binary field called charted indicating whether a song reached the Billboard charts or not.

```
#create binary field called charted showing if a song hit the charts at all
prince <- prince %>%
  mutate(charted =
           ifelse(prince$peak %in% 1:100, "Charted", "Uncharted"))

#save the new dataset to .csv for use in later tutorials
write.csv(prince, file = "prince_new.csv")
```

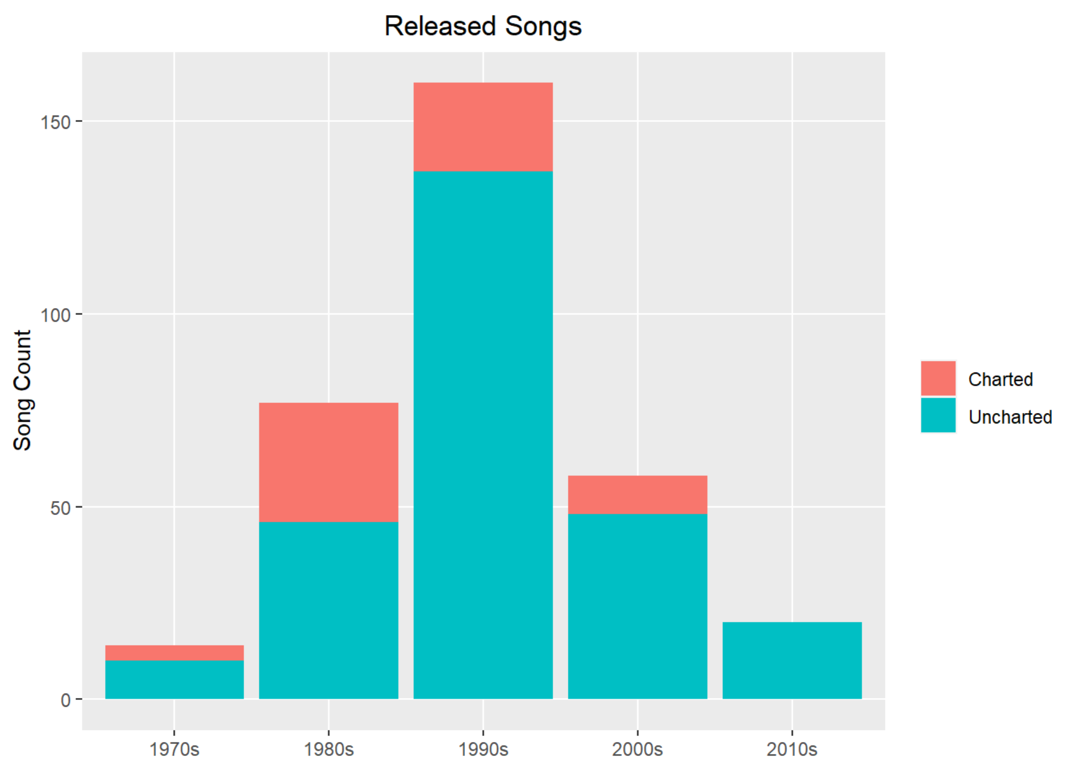
DESCRIPTIVE ANALYTICS

```
#define some colors to use throughout
my_colors <- c("#E69F00", "#56B4E9", "#009E73", "#CC79A7", "#D55E00")

theme_lyrics <- function()
{
  theme(plot.title = element_text(hjust = 0.5),
        axis.text.x = element_blank(),
        axis.ticks = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        legend.position = "none")
}
```

As a reminder, prince professional career started in 1978 and went all the way through 2015 (as shown in the summary() statistics above). But since we’re looking at trends now, and the dataset has a large number of blank values for year, you’ll want to filter out all the release year NAs for the first graph. Using dplyr’s filter(),group_by() and summarise() functions, can group by decade and then count the number of songs. The function n() is one of several aggregate functions that is useful to employ with summarise() on grouped data. Then using ggplot() and geom_bar(), create a bar chart and fill the bars with the charted category.

```
prince %>%
  filter(decade != "NA") %>%
  group_by(decade, charted) %>%
  summarise(number_of_songs = n()) %>%
  ggplot() +
  geom_bar(aes(x = decade, y = number_of_songs,
               fill = charted), stat = "identity")  +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        panel.grid.minor = element_blank()) +
  ggtitle("Released Songs") +
  labs(x = NULL, y = "Song Count")
```

```
## `summarise()` regrouping output by 'decade' (override with `.groups` argument)
```

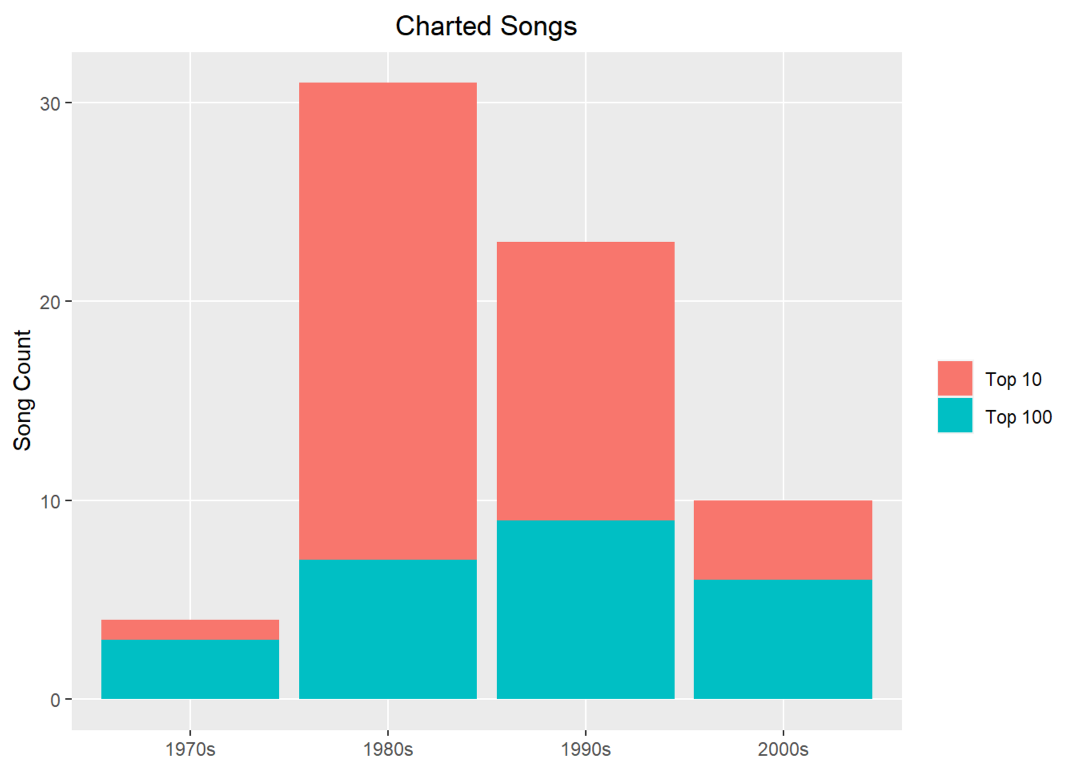


This clearly shows his most active decade was the 1990s.Now create a similar graph with chart_level.group_by() both decade and chart_level so you can see the trend.

```
charted_songs_over_time <- prince %>%
  filter(peak > 0) %>%
  group_by(decade, chart_level) %>%
  summarise(number_of_songs = n())
```

```
## `summarise()` regrouping output by 'decade' (override with `.groups` argument)
```

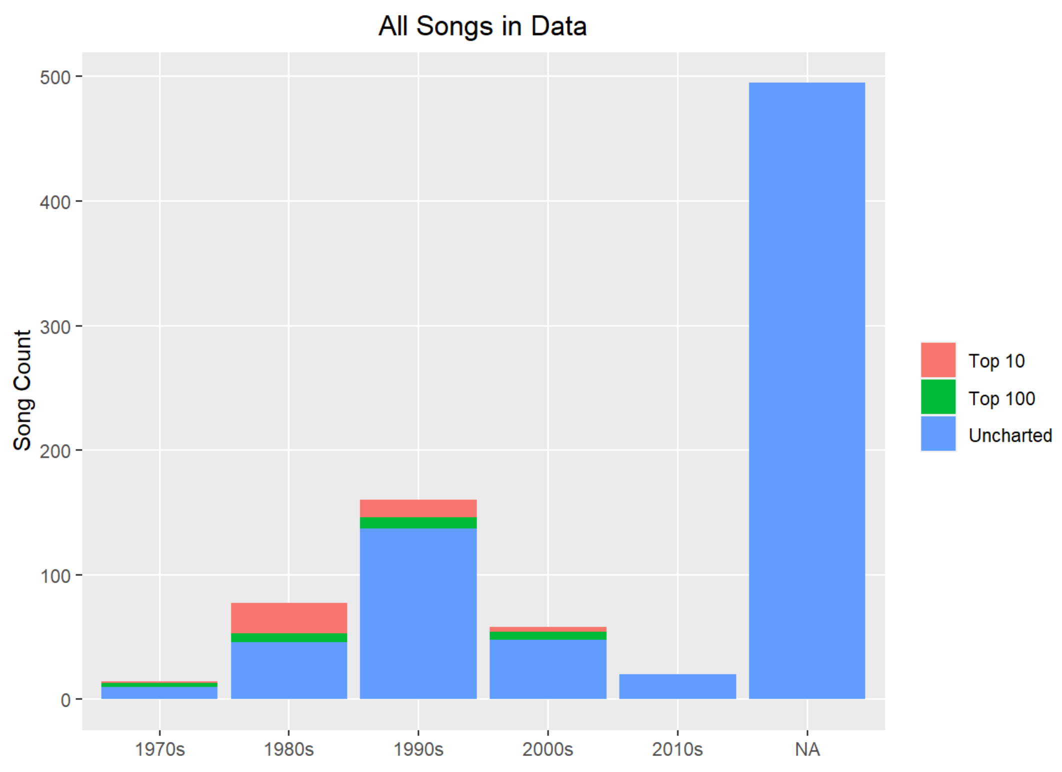
```
charted_songs_over_time %>%
  ggplot() +
  geom_bar(aes(x = decade, y = number_of_songs,
               fill = chart_level), stat = "identity") +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        panel.grid.minor = element_blank()) +
  labs(x = NULL, y = "Song Count") +
  ggtitle("Charted Songs")
```



Notice that out of all Prince's charted songs, the majority reached Top 10. But what is even more interesting is that his most prolific decade for new songs was the 1990s, yet more chart hits occurred in the 1980s. In order to use the power of the full dataset for lyrical analysis, you can remove references to chart level and release year.

```
#look at the full data set
prince %>%
  group_by(decade, chart_level) %>%
  summarise(number_of_songs = n()) %>%
  ggplot() +
  geom_bar(aes(x = decade, y = number_of_songs,
               fill = chart_level), stat = "identity")  +
  theme(plot.title = element_text(hjust = 0.5),
        legend.title = element_blank(),
        panel.grid.minor = element_blank()) +
  labs(x = NULL, y = "Song Count") +
  ggtitle("All Songs in Data")
```

```
## `summarise()` regrouping output by 'decade' (override with `.groups` argument)
```



As you can see, Prince wrote hundreds of songs for which there is no release date in the data. #NO.1 SONGS

```
prince %>%
  filter(peak == "1") %>%
  select(year, song, peak) %>%
  arrange(year) %>%
  mutate(year = color_tile("lightblue", "lightgreen")(year)) %>%
  mutate(peak = color_tile("lightgreen", "lightgreen")(peak)) %>%
  kable("html", escape = FALSE, align = "c", caption = "Prince's No. 1 Songs") %>%
  kable_styling(bootstrap_options =
                c("striped", "condensed", "bordered"),
                full_width = FALSE)
```

Prince’s No. 1 Songs

year	song	peak
1979	i wanna be your lover	1
1984	erotic city	1
1984	purple rain	1
1984	when doves cry	1
1985	around the world in a day	1
1986	kiss	1
1988	lovesexy	1
1989	batdance	1
1990	thieves in the temple	1
1991	diamonds and pearls	1
1995	the most beautiful girl in the world	1
2006	3121	1
2007	planet earth	1

##TEXT MINING Text mining can also be thought of as text analytics. The goal is to discover relevant information that is possibly unknown or buried beneath the obvious. Natural Language Processing (NLP) is one methodology used in mining text. It tries to decipher the ambiguities in written language by tokenization, clustering, extracting entity and word relationships, and using algorithms to identify themes and quantify subjective information. Lexical complexity can mean different things in different contexts. Word Frequency: number of words per song Word Length: average length of individual words in a text Lexical Diversity: number of unique words used in a text (song vocabulary) Lexical Density: the number of unique words divided by the total number of words (word repetition) Breaking out the lyrics into individual words and begin mining for insights and this process is called tokenization. There are different methods and data formats that can be used to mine text: Corpus: a collection of documents that are created by the tm text mining package Document-term matrix: a matrix that lists all occurrences of words in the corpus, by document, where documents are rows and words are columns Tidy Text: a table with one token per row. Tokenization is therefore the process of splitting the lyrics into tokens. List of superfluous words that need to be removed manually before text mining is performed.

```
undesirable_words <- c("prince", "chorus", "repeat", "lyrics",
                       "theres", "bridge", "fe0f", "yeah", "baby",
                       "alright", "wanna", "gonna", "chorus", "verse",
                       "whoa", "gotta", "make", "miscellaneous", "2",
                       "4", "ooh", "uurh", "pheromone", "poompoom", "3121",
                       "matic", " ai ", " ca ", " la ", "hey", " na ",
                       " da ", " uh ", " tin ", "  ll", "transcription",
                       "repeats")
```

To unnest the tokens, use the tidytext library, which has already been loaded.unnest_tokens() requires at least two arguments: the output column name that will be created as the text is unnested into it ("word", in this case), and the input column that holds the current text (i.e. lyrics). Prince dataset and pipe it into unnest_tokens() and then remove stop words.Stop words are overly common words that may not add any meaning to our results. Use the lexicon called stop_words from the tidytext package.Use sample() to show a random list of these stop words and head() to limit to 15 words.

```
head(sample(stop_words$word, 15), 15)
```

```
##  [1] "because"   "how"       "needed"    "however"   "quite"     "anyway"
##  [7] "between"   "together"  "a"         "neither"   "self"      "important"
## [13] "all"       "took"      "full"
```

Tokenize the lyrics into words, you can then use dplyr's anti_join() to remove stop words. Use distinct() to get rid of any duplicate records as well.you can remove all words with fewer than four characters. This is another subjective decision, but in lyrics, these are often interjections such as "yea and hey". Then store the results into prince_words_filtered.prince_words_filtered is the tidy text version of the prince data frame without 1) stop words, 2) undesirable words, and 3) 1-3 character words.

```
#unnest and remove stop, undesirable and short words
prince_words_filtered <- prince %>%
  unnest_tokens(word, lyrics) %>%
  anti_join(stop_words) %>%
  distinct() %>%
  filter(!word %in% undesirable_words) %>%
  filter(nchar(word) > 3)
```

```
## Joining, by = "word"
```

Notice that stop_words has a word column, and a new column called word was created by the unnest_tokens() function, so anti_join() automatically joins on the column word.

```
class(prince_words_filtered)
```

```
## [1] "data.frame"
```

```
dim(prince_words_filtered)
```

```
## [1] 36916    10
```

```
prince_words_filtered %>%
  filter(word == "race") %>%
  select(word, song, year, peak, decade, chart_level, charted) %>%
  arrange() %>%
  top_n(10,song) %>%
  mutate(song = color_tile("lightblue","lightblue")(song)) %>%
  mutate(word = color_tile("lightgreen","lightgreen")(word)) %>%
  kable("html", escape = FALSE, align = "c", caption = "Tokenized Format Example") %>%
  kable_styling(bootstrap_options =
                c("striped", "condensed", "bordered"),
                full_width = FALSE)
```

```
## Warning: Problem with `mutate()` input `song`.
## i NAs introduced by coercion
## i Input `song` is `color_tile("lightblue", "lightblue")(song)`.
```

```
## Warning in gradient(as.numeric(x), ...): NAs introduced by coercion
```

```
## Warning: Problem with `mutate()` input `word`.
## i NAs introduced by coercion
## i Input `word` is `color_tile("lightgreen", "lightgreen")(word)`.
```

```
## Warning in gradient(as.numeric(x), ...): NAs introduced by coercion
```

Tokenized Format Example

word	song	year	peak	decade	chart_level	charted
race	lovesexy	1988	1	1980s	Top 10	Charted
race	my tree	NA	NA	NA	Uncharted	Uncharted
race	positivity	1988	NA	1980s	Uncharted	Uncharted
race	race	1994	NA	1990s	Uncharted	Uncharted
race	sexuality	1981	88	1980s	Top 100	Charted
race	slow love	1987	NA	1980s	Uncharted	Uncharted
race	the rest of my life	1999	NA	1990s	Uncharted	Uncharted
race	the undertaker	NA	NA	NA	Uncharted	Uncharted
race	u make my sun shine	NA	NA	NA	Uncharted	Uncharted
race	welcome 2 the rat race	NA	NA	NA	Uncharted	Uncharted

Each row contains an individual word which would be repeated for each song in which it appears. #WORD FREQUENCY Individual word frequencies carry a great deal of importance, whether it be repetition or rarity.

```
full_word_count <- prince %>%
  unnest_tokens(word, lyrics) %>%
  group_by(song,chart_level) %>%
  summarise(num_words = n()) %>%
  arrange(desc(num_words))
```

```
## `summarise()` regrouping output by 'song' (override with `.groups` argument)
```

```
full_word_count[1:10,] %>%
  ungroup(num_words, song) %>%
  mutate(num_words = color_bar("lightblue")(num_words)) %>%
  mutate(song = color_tile("lightpink","lightpink")(song)) %>%
  kable("html", escape = FALSE, align = "c", caption = "Songs With Highest Word Count") %>%
  kable_styling(bootstrap_options =
                c("striped", "condensed", "bordered"),
                full_width = FALSE)
```

```
## Warning: Problem with `mutate()` input `song`.
## i NAs introduced by coercion
## i Input `song` is `color_tile("lightpink", "lightpink")(song)`.
```

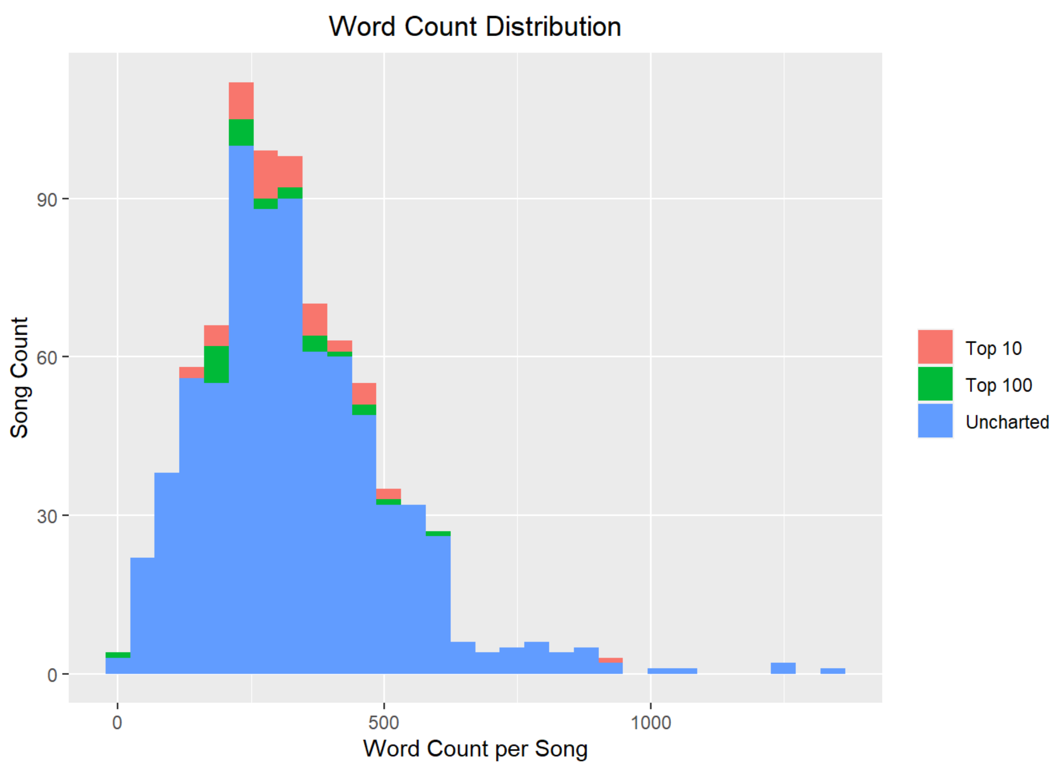
```
## Warning in gradient(as.numeric(x), ...): NAs introduced by coercion
```

Songs With Highest Word Count

song	chart_level	num_words
johnny	Uncharted	1349
cloreen bacon skin	Uncharted	1263
push it up	Uncharted	1240
the exodus has begun	Uncharted	1072
wild and loose	Uncharted	1031
jughead	Uncharted	940
my name is prince	Top 10	916
acknowledge me	Uncharted	913
the walk	Uncharted	883
the purple medley	Uncharted	874

```r
#WORD COUNT DISTRIBUTION PLOT
full_word_count %>%
  ggplot() +
    geom_histogram(aes(x = num_words, fill = chart_level )) +
    ylab("Song Count") +
    xlab("Word Count per Song") +
    ggtitle("Word Count Distribution") +
    theme(plot.title = element_text(hjust = 0.5),
          legend.title = element_blank(),
          panel.grid.minor.y = element_blank())
```

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



Given the nature of the lyric transcription, I'm skeptical about data entry errors.

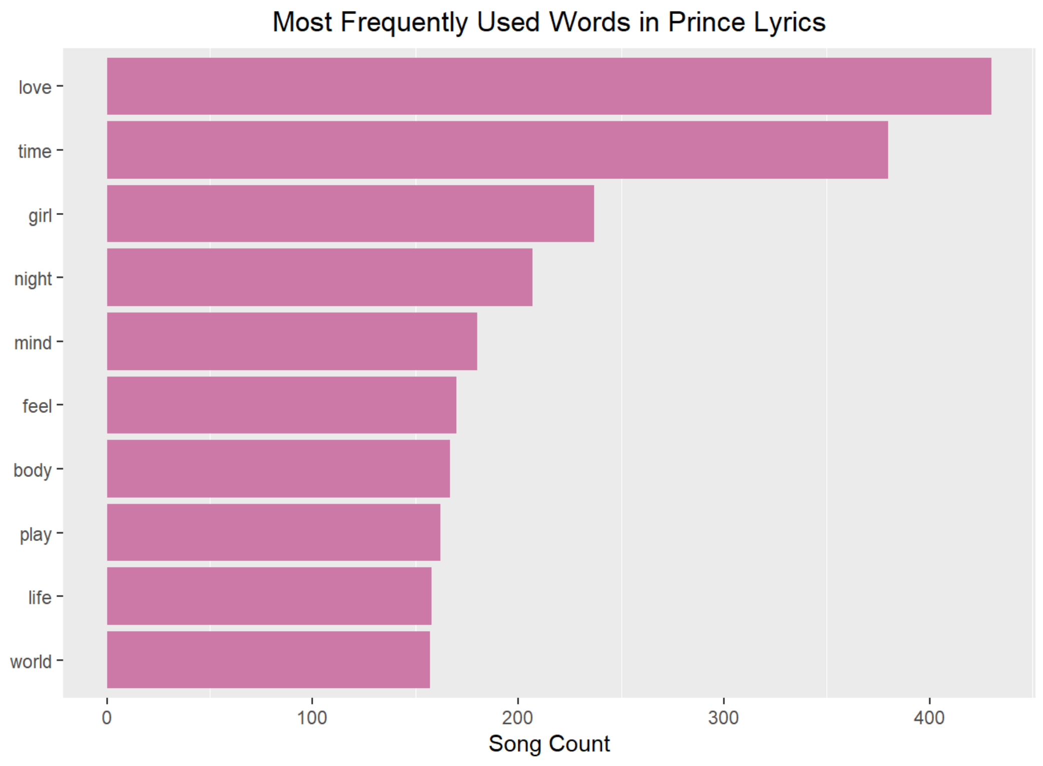
```
full_word_count %>%
  filter(chart_level == 'Top 10' & num_words > 800) %>%
  left_join(prince_orignal, by = "song") %>%
  select(Song = song,
         "Word Count" = num_words,
         "Peak Position" = peak,
         "US Pop" = US.Pop,
         "US R&B" = US.R.B,
         Canada = CA,
         Ireland = IR) %>%
  kable("html", escape = FALSE) %>%
  kable_styling(bootstrap_options = c("striped", "condensed", "bordered"))
```

Song	Word Count	Peak Position	US Pop	US R&B	Canada	Ireland
my name is prince	916	5	36	25	5	5

#Top Words In order to do a simple evaluation of the most frequently used words in the full set of lyrics, you can use count() and top_n() to get the n top words from your clean, filtered dataset. Then use reorder() to sort words according to the count and use dplyr's mutate() verb to reassign the ordered value to word. This allows ggplot() to display it nicely.

```
prince_words_filtered %>%
  count(word, sort = TRUE) %>%
  top_n(10) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot() +
    geom_col(aes(word, n), fill = my_colors[4]) +
    theme(legend.position = "none",
          plot.title = element_text(hjust = 0.5),
          panel.grid.major = element_blank()) +
    xlab("") +
    ylab("Song Count") +
    ggtitle("Most Frequently Used Words in Prince Lyrics") +
    coord_flip()
```

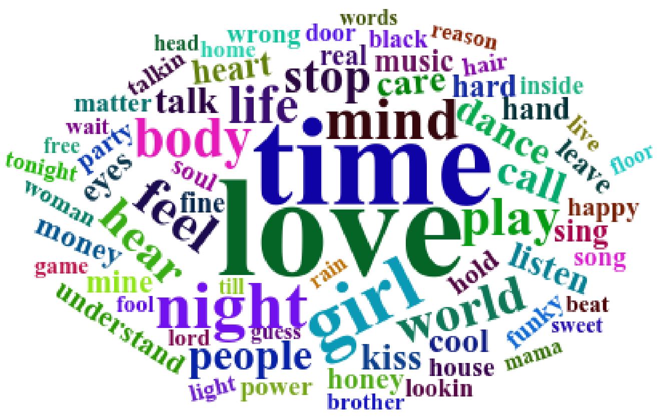
```
## Selecting by n
```



As in most popular music, love seems to be a common topic.

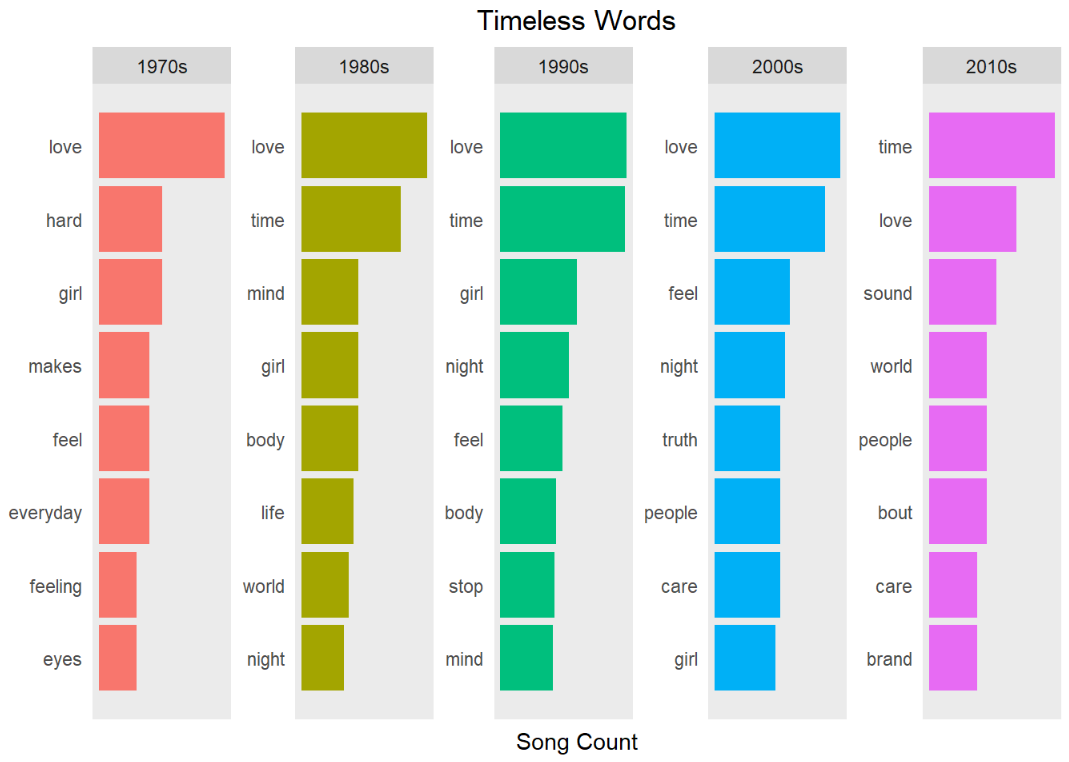
#Word Clouds Word clouds get a bad rap in many circles, and if you're not careful with them, they can be used out of context where they are not appropriate.wordcloud2 package gives you a creative collection of clouds that generate html widgets.

```
prince_words_counts <- prince_words_filtered %>%
  count(word, sort = TRUE)

wordcloud2(prince_words_counts[1:300, ], size = .5)
```



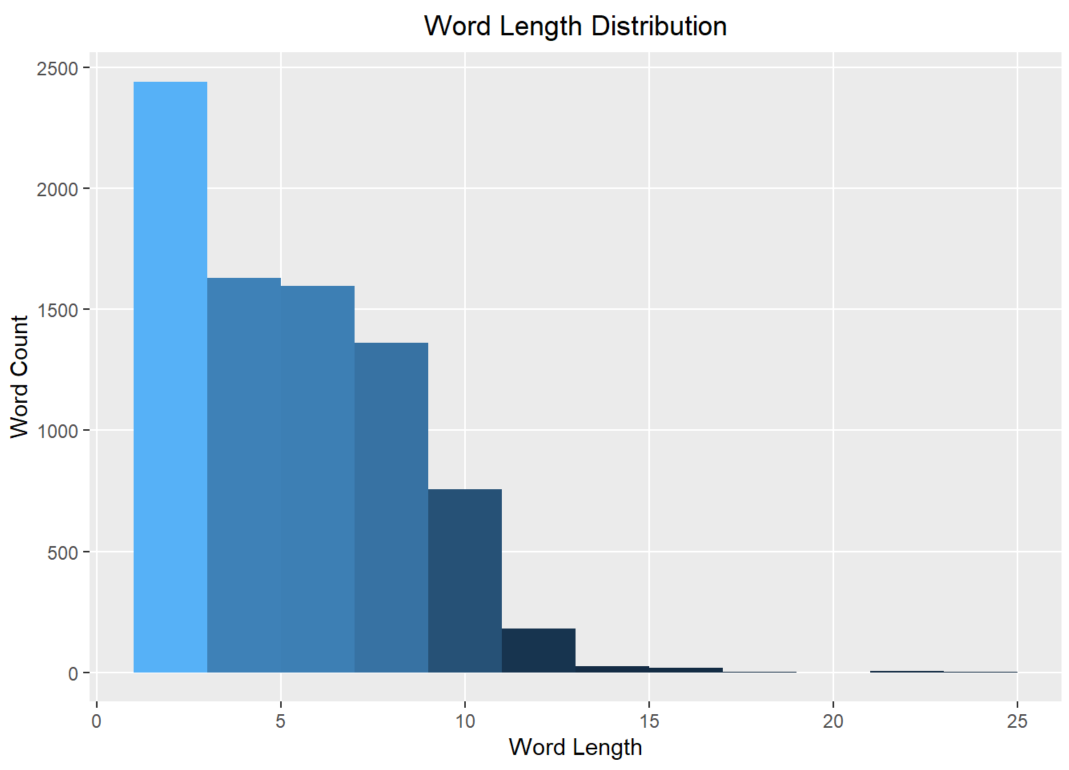
Some words in music are considered to be timeless. Timeless words persevere over time, appealing to a large audience.

```
timeless_words <- prince_words_filtered %>%
  filter(decade != 'NA') %>%
  group_by(decade) %>%
  count(word, decade, sort = TRUE) %>%
  slice(seq_len(8)) %>%
  ungroup() %>%
  arrange(decade,n) %>%
  mutate(row = row_number())

timeless_words %>%
  ggplot(aes(row, n, fill = decade)) +
    geom_col(show.legend = NULL) +
    labs(x = NULL, y = "Song Count") +
    ggtitle("Timeless Words") +
    theme_lyrics() +
    facet_wrap(~decade, scales = "free", ncol = 5) +
    scale_x_continuous(  # This handles replacement of row
      breaks = timeless_words$row, # notice need to reuse data frame
      labels = timeless_words$word) +
    coord_flip()
```



Clearly love, time, and girl are timeless. #Word Length Word length is an interesting topic for songwriters. The longer the word, the harder it is to rhyme and squeeze into a pattern. #unnest and remove undesirable words, but leave in stop and short words

```
prince_word_lengths <- prince %>%
  unnest_tokens(word, lyrics) %>%
  group_by(song,decade) %>%
  distinct() %>%
  filter(!word %in% undesirable_words) %>%
  mutate(word_length = nchar(word))

prince_word_lengths %>%
  count(word_length, sort = TRUE) %>%
  ggplot(aes(word_length),
         binwidth = 10) +
    geom_histogram(aes(fill = ..count..),
                   breaks = seq(1,25, by = 2),
                   show.legend = FALSE) +
    xlab("Word Length") +
    ylab("Word Count") +
    ggtitle("Word Length Distribution") +
    theme(plot.title = element_text(hjust = 0.5),
          panel.grid.minor = element_blank())
```



```
wc <- prince_word_lengths %>%
  ungroup() %>%
  select(word, word_length) %>%
  distinct() %>%
  arrange(desc(word_length))

wordcloud2(wc[1:300, ],
           size = .15,
           minSize = .0005,
           ellipticity = .3,
           rotateRatio = 1,
           fontWeight = "bold")
```

#Lexical Diversity The more varied a vocabulary a text possesses, the higher its lexical diversity. Song Vocabulary is a representation of how many unique words are used in a song.

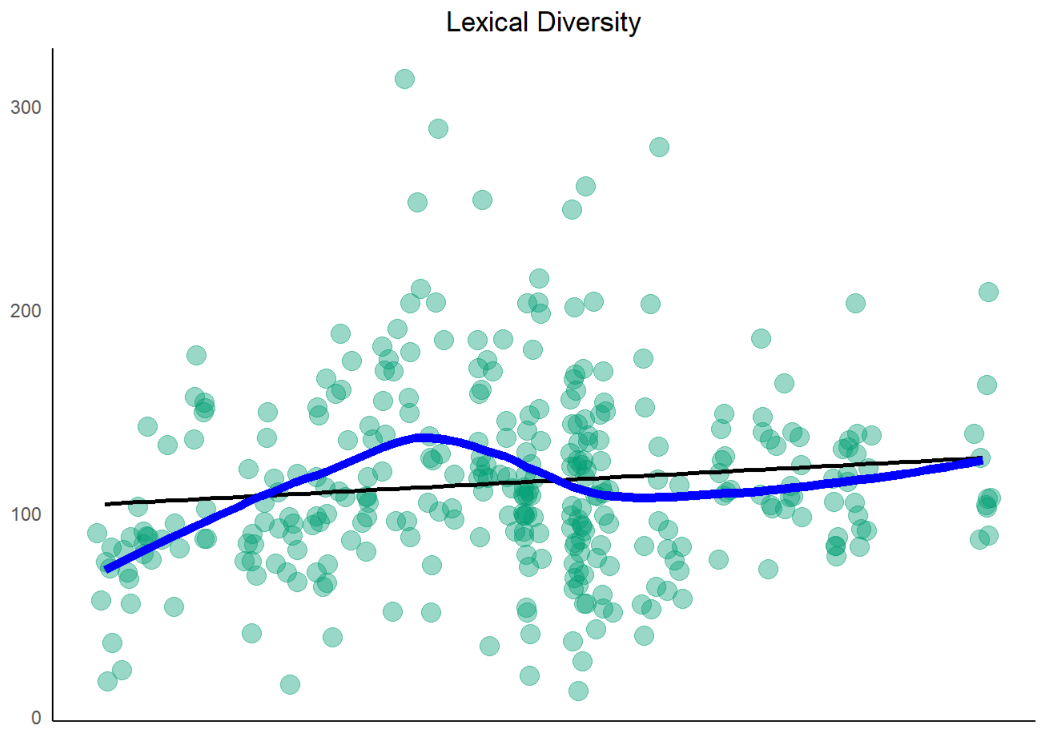
```
lex_diversity_per_year <- prince %>%
  filter(decade != "NA") %>%
  unnest_tokens(word, lyrics) %>%
  group_by(song,year) %>%
  summarise(lex_diversity = n_distinct(word)) %>%
  arrange(desc(lex_diversity))
```

```
## `summarise()` regrouping output by 'song' (override with `.groups` argument)
```

```
diversity_plot <- lex_diversity_per_year %>%
  ggplot(aes(year, lex_diversity)) +
    geom_point(color = my_colors[3],
               alpha = .4,
               size = 4,
               position = "jitter") +
    stat_smooth(color = "black", se = FALSE, method = "lm") +
    geom_smooth(aes(x = year, y = lex_diversity), se = FALSE,
                color = "blue", lwd = 2) +
    ggtitle("Lexical Diversity") +
    xlab("") +
    ylab("") +
    scale_color_manual(values = my_colors) +
    theme_classic() +
    theme_lyrics()

diversity_plot
```

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

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



Over the past few decades, there was a slight upward trend in Prince’s lyric diversity. How does this correlate to chart success? Hard to say, really, but go ahead and plot density and chart history for further comparison. #Lexical Density Lexical density is defined as the number of unique words divided by the total number of words. This is an indicator of word repetition, which is a critical songwriter’s tool. Group by song and year and use n_distinct() and n() to calculate the density. Pipe that into ggplot() with geom_smooth(). Add an additional stat_smooth() with method=“lm” for a linear smooth model.

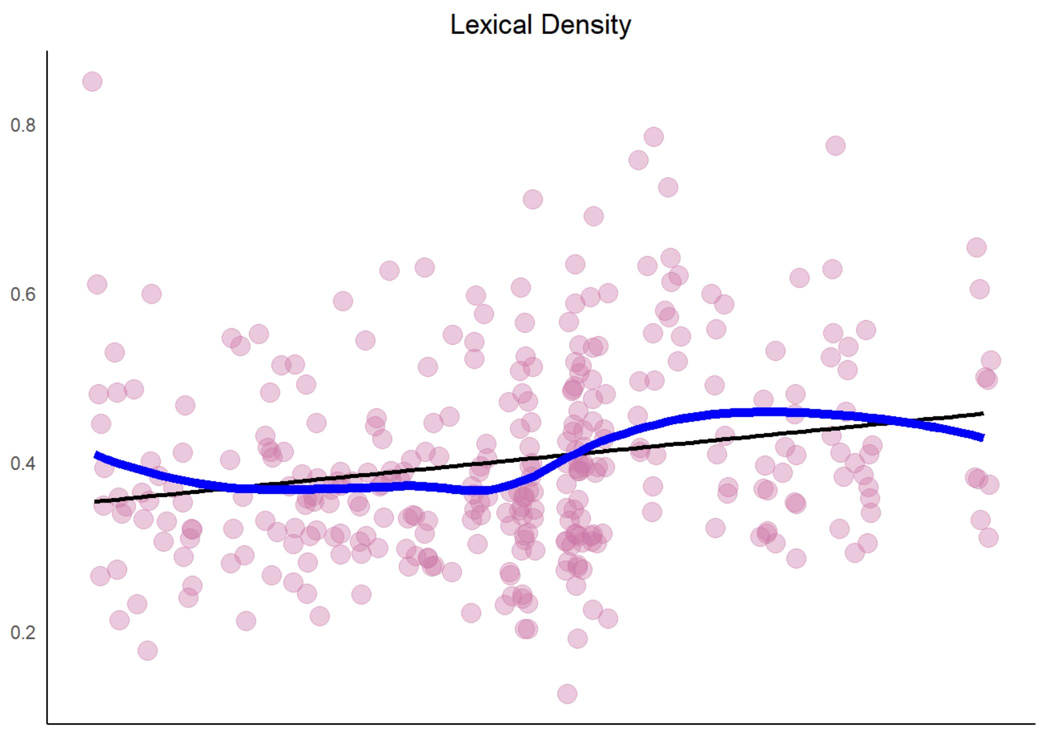
```
 lex_density_per_year <- prince %>%
  filter(decade != "NA") %>%
  unnest_tokens(word, lyrics) %>%
  group_by(song,year) %>%
  summarise(lex_density = n_distinct(word)/n()) %>%
  arrange(desc(lex_density))
```

```
## `summarise()` regrouping output by 'song' (override with `.groups` argument)
```

```
density_plot <- lex_density_per_year %>%
  ggplot(aes(year, lex_density)) +
    geom_point(color = my_colors[4],
               alpha = .4,
               size = 4,
               position = "jitter") +
    stat_smooth(color = "black",
                se = FALSE,
                method = "lm") +
    geom_smooth(aes(x = year, y = lex_density),
                se = FALSE,
                color = "blue",
                lwd = 2) +
    ggtitle("Lexical Density") +
    xlab("") +
    ylab("") +
    scale_color_manual(values = my_colors) +
    theme_classic() +
    theme_lyrics()

density_plot
```

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

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



In order to compare the trends, create a plot for his chart history (i.e. successful songs that reached the charts) and compare it to diversity and density.correlation doesn’t imply causation

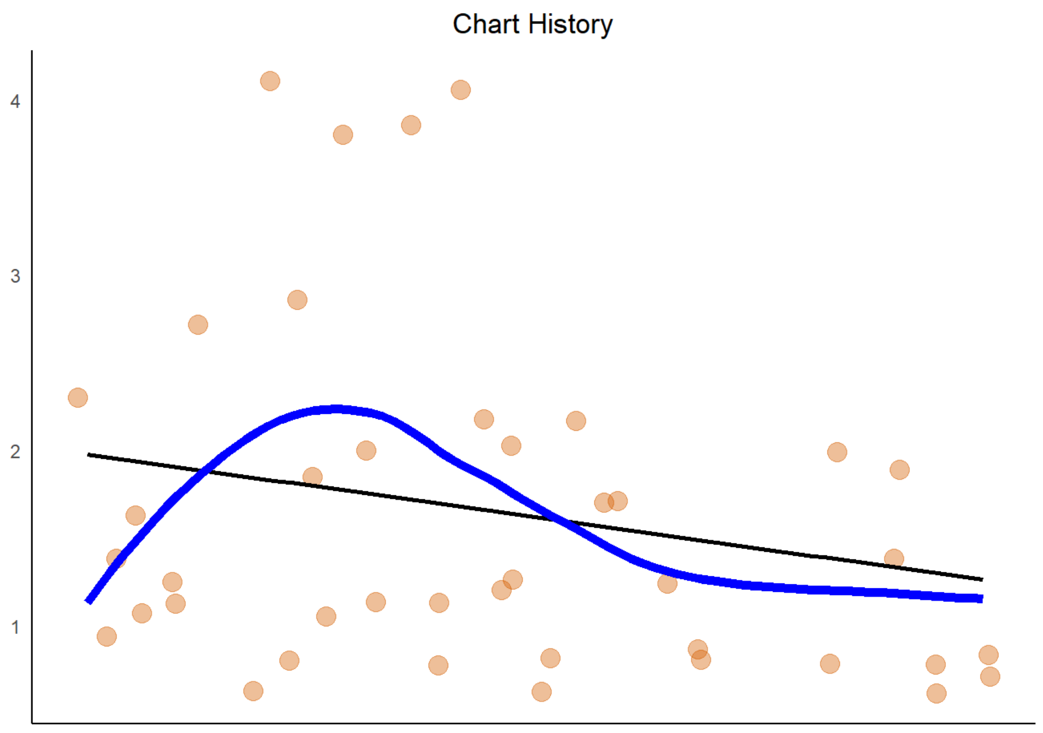
```
chart_history <- prince %>%
  filter(peak > 0) %>%
  group_by(year, chart_level) %>%
  summarise(number_of_songs = n()) %>%
  ggplot(aes(year, number_of_songs)) +
    geom_point(color = my_colors[5],
               alpha = .4,
               size = 4,
               position = "jitter") +
    geom_smooth(aes(x = year, y = number_of_songs),
                se = FALSE,
                method = "lm",
                color = "black" ) +
    geom_smooth(aes(x = year, y = number_of_songs),
                se = FALSE,
                color = "blue",
                lwd = 2) +
   ggtitle("Chart History") +
   xlab("") +
   ylab("") +
   scale_color_manual(values = my_colors) +
   theme_classic() +
   theme_lyrics()
```

```
## `summarise()` regrouping output by 'year' (override with `.groups` argument)
```

```
chart_history
```

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

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



#TF-IDF TF is “term

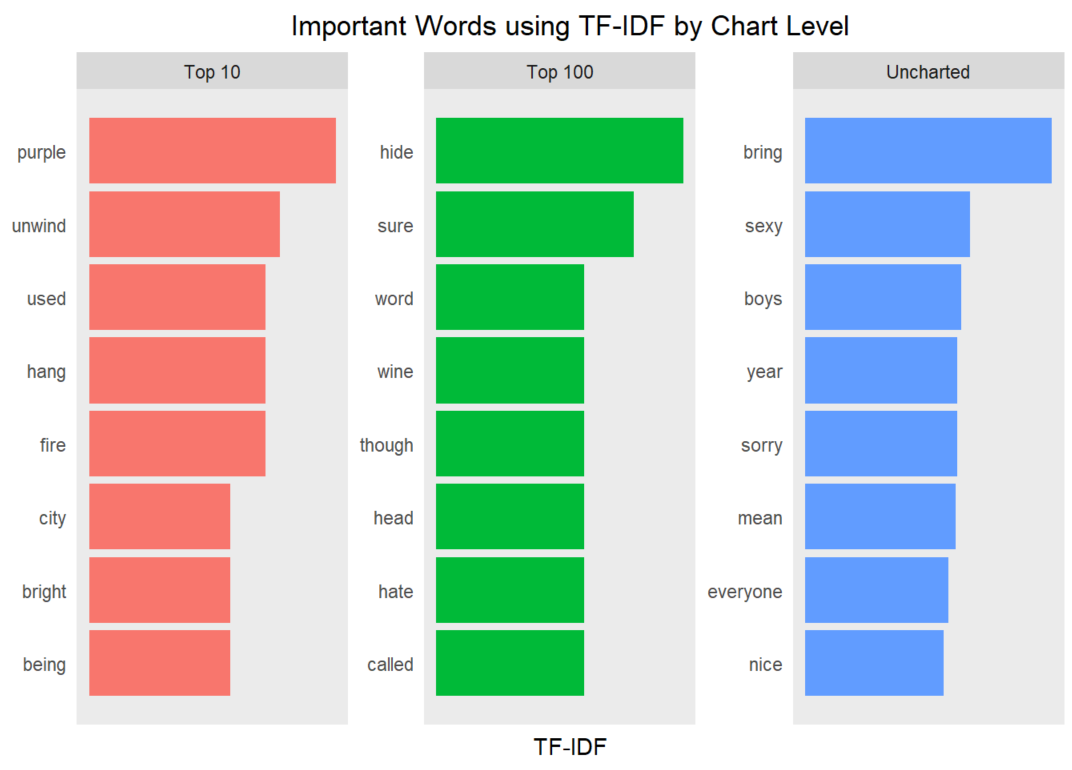
frequency”. IDF is “inverse document frequency”, which attaches a lower weight for commonly used words and a higher weight for words that are not used much in a collection of text. When you combine TF and IDF, a term’s importance is adjusted for how rarely it is used. The assumption behind TF-IDF is that terms that appear more frequently in a document should be given a higher weight, unless it also appears in many documents. The formula can be summarized below:

Term Frequency (TF): Number of times a term occurs in a document Document Frequency (DF): Number of documents that contain each word Inverse Document Frequency (IDF) = 1/DF TF-IDF = TF * IDF This new approach to examine the most important words per chart level with the bind_tf_idf() function provided by tidytext. This function calculates and binds the TF and IDF, along with the TF*IDF product. It takes a tidy text dataset as input with one row per token (word), per document (song).

```
popular_tfidf_words <- prince %>%
  unnest_tokens(word, lyrics) %>%
  distinct() %>%
  filter(!word %in% undesirable_words) %>%
  filter(nchar(word) > 3) %>%
  count(chart_level, word, sort = TRUE) %>%
  ungroup() %>%
  bind_tf_idf(word, chart_level, n)

head(popular_tfidf_words)
```

TF-IDF are 0 for extremely common words. (Technically, the IDF term will be the natural log of 1 and thus would be zero for these words.)

```
top_popular_tfidf_words <- popular_tfidf_words %>%
  arrange(desc(tf_idf)) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(chart_level) %>%
  slice(seq_len(8)) %>%
  ungroup() %>%
  arrange(chart_level, tf_idf) %>%
  mutate(row = row_number())

top_popular_tfidf_words %>%
  ggplot(aes(x = row, tf_idf,
             fill = chart_level)) +
    geom_col(show.legend = NULL) +
    labs(x = NULL, y = "TF-IDF") +
    ggtitle("Important Words using TF-IDF by Chart Level") +
    theme_lyrics() +
    facet_wrap(~chart_level, ncol = 3, scales = "free") +
    scale_x_continuous(  # This handles replacement of row
      breaks = top_popular_tfidf_words$row, # notice need to reuse data frame
      labels = top_popular_tfidf_words$word) +
    coord_flip()
```



##Conclusion After performing some conditioning such as data cleansing and removing uninformative words, you began an exploratory analysis at the song level. The results provide critical insights for the next steps of sentiment analysis and topic modeling. Finally used TF-IDF analysis to represent the information behind a word in a document relating to some outcome of interest.