Report 1

Introduction

I listen to a lot of music. I've never really had a musical bone in my body so I tend to gravitate toward the lyrics of a song (which is a reason why you'll never find an EDM song in my playlists). After finding a dataset on Kaggle containing the lyrics of Billboard's Hot100 year-end chart from 1964-2015, I decided that it would be interesting to use the lyrics to try to predict a given song's success on the BillBoard Hot100. The Hot100 is just a list of the most popular songs of the given year ranked from 1 (most popular) to 100 (100-most popular).

Cleaning the data

Here's what my original data set looked like.

```
original_data = read.csv("billboard_lyrics_1964-2015.csv")
names(original_data)
```

```
## [1] "Rank"   "Song"   "Artist" "Year"   "Lyrics" "Source"
```

```
# I don't know how to stop the lyrics from messing it up
head(data)
```

```
##
## 1 function (..., list = character(), package = NULL, lib.loc = NULL,
## 2     verbose = getOption("verbose"), envir = .GlobalEnv)
## 3 {
## 4     fileExt <- function(x) {
## 5         db <- grepl("\\\\.[^.]+\\\\.(gz|bz2|xz)$", x)
## 6         ans <- sub(".*\\\\.", "", x)
```

I decided that I wanted to come up with more columns to better interpret lyrics and had to come up with some independet variables on my own. I wrote a python script to help me. The script, which is below, imports the lyrics from the data set and finds the number of words, average word length, average rhyme length, and number of unique words in the given lyrics.

(I would suggest not running this.)

```
import nltk
import sys,os
import pandas as pd
import phonetics as ph
import lyrics as ly

# nltk.download("cmudict")
data = pd.read_csv("billboard_lyrics_1964-2015.csv", encoding = "ISO-8859-1")
# data = pd.read_csv("data_new.csv")

def get_unique_words(lyrics):
    words = lyrics.split(sep=" ")
```

```
    unique_words = dict()
    sum = 0
    for word in words :
        sum += len(word)
        if word not in unique_words:
            unique_words[word] = 1
        else:
            unique_words[word] += 1
    avg = sum / len(words)
    if len(words) is 0 :
        length = 0
    else:
        length = len(words)
    return(len(unique_words),avg,length)



'''
This finds the average rhyme length of every song in the table and binds it to
the original data. Saves new data table to new csv.

uses Eric Malme's algorithm to measure the average length of rhymes in lyrics.
github: https://github.com/ekQ/raplysaattori
'''
avg_rhyme_length = []
unique_words = []
average_lengths = []
num_words = []

sys.stdout = open(os.devnull, 'w')

for i in range(0,len(data.index)) :
    # sometimes the lyrics are float values, idk why
    # if the lyrics aren't a string, the avg_rhyme_length is set to a constant 9999
    if type(data.loc[i].Lyrics) is not type(" ") or data.loc[i].Lyrics is "" :
        avg_rhyme_length.append(9999)
        unique_words.append(9999)
        average_lengths.append(9999)
        num_words.append(9999)
    else :
        # clean lyrics
        lyric = data.loc[i].Lyrics.strip()
        lyric = lyric.replace("  "," ")

        w,a,n = get_unique_words(lyric)
        unique_words.append(w)
        average_lengths.append(a)
        num_words.append(n)

        l = ly.Lyrics(print_stats=None,text=lyric,language="en",lookback=15)
        avg_rhyme_length.append(l.get_avg_rhyme_length())

columns = pd.DataFrame({
```

```
                'AverageRhymeLength': avg_rhyme_length,
                'UniqueWords' : unique_words,
                'AverageLengths' : average_lengths,
                'NumberWords' : num_words})

# columns = pd.DataFrame(unique_words,columns=['unique_words'])
data_new = pd.concat([data,columns],axis=1,ignore_index=False)
data_new.to_csv("data_new.csv",index=True)

sys.stdout = sys.__stdout__
print("saved")
```

After saving as a csv, I imported the new data frame into R and used a package called SentimentAnalysis to calculate a sentiment score for each of the lyrics. Here's the code:

```
data = read.csv("data_new.csv")
data = data %>%
  mutate("Sentiment" = analyzeSentiment(as.character(Lyrics))$SentimentQDAP) %>%
  rename("AverageWordLength" = AverageLengths)

write.csv(data,"data_clean.csv")
```

Here are the Xs broken down:

1. **Number of words** - This one should be pretty self-explanatory: How many words (non-unique included) does the song contain?
2. **Average Word Length** - Does the songwriter use lengthier words or smaller words? The script gets this by adding all the characters in the lyrics and dividing by the total number of words in the lyrics.
3. **Average Rhyme Length** - I had to use a handy library I found to compute this one. I'll link the github at the end, but here's a quick rundown of how it works:

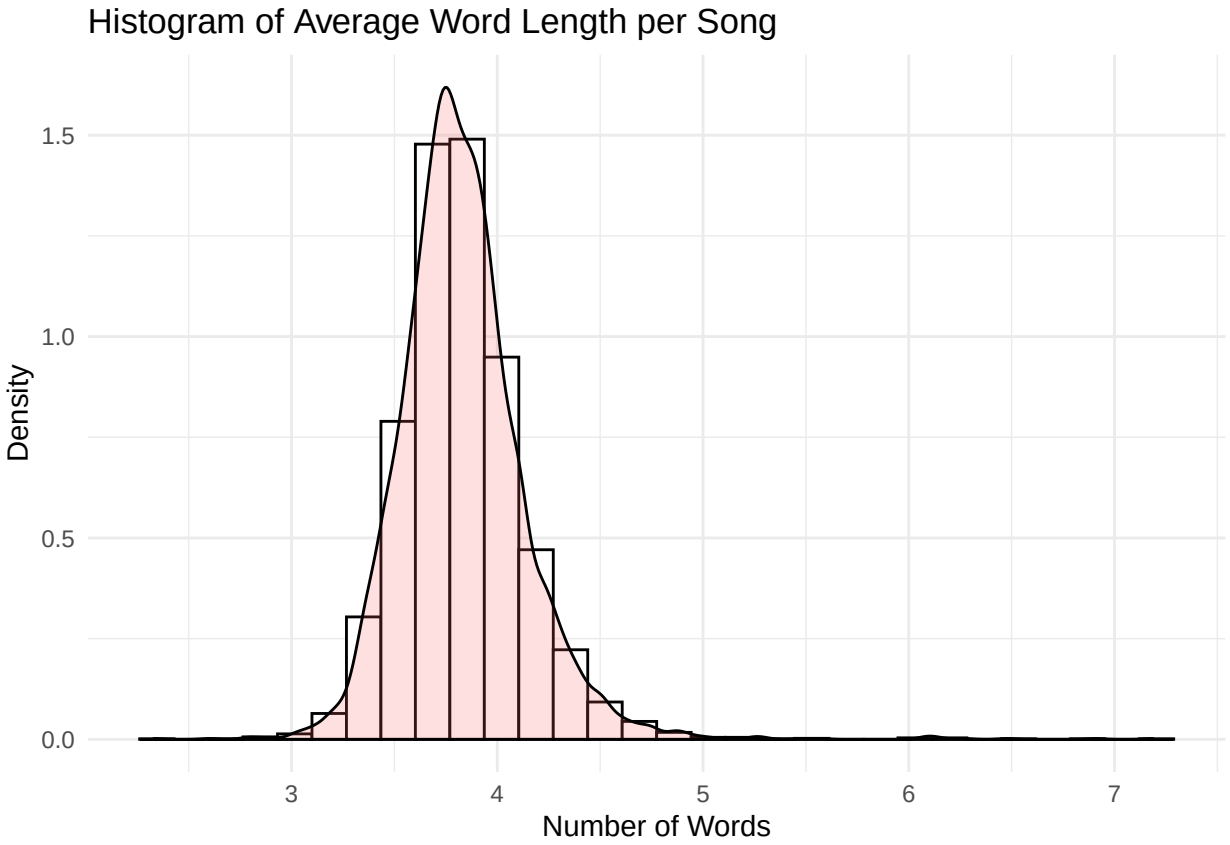
- Go through lyrics word for word, using eSpeak to get the phonetic make-up of each word.
- For every word, find the longest matching rhyme senquence from the last 15 words. This is what captures the length of the rhyme. Since eSpeak converts all the words to phonetics (and I think just vowel phonetics) only vowels are compared, which is what gives the program something to match. Essentially, the program searches for the longest matching vowel sequence for last X words, where I used an abitrary 15 words for X.
- Find the average rhyme length by summing all the lengths and dividing by the number of rhymes. Here's the github

4. **Number of Unique Words** - I used python's dictionaries to keep track of the number of unique words in each song. Easy enough.
5. **Sentiment** - I'm not entirely sure what algorithm is used in this package, but I used the analyzeSentiment function from SentimentAnalysis to get each number. The sentiment is a score that gives the general "emotion" score of a word or series of words from -1 to 1.

Examing the Data

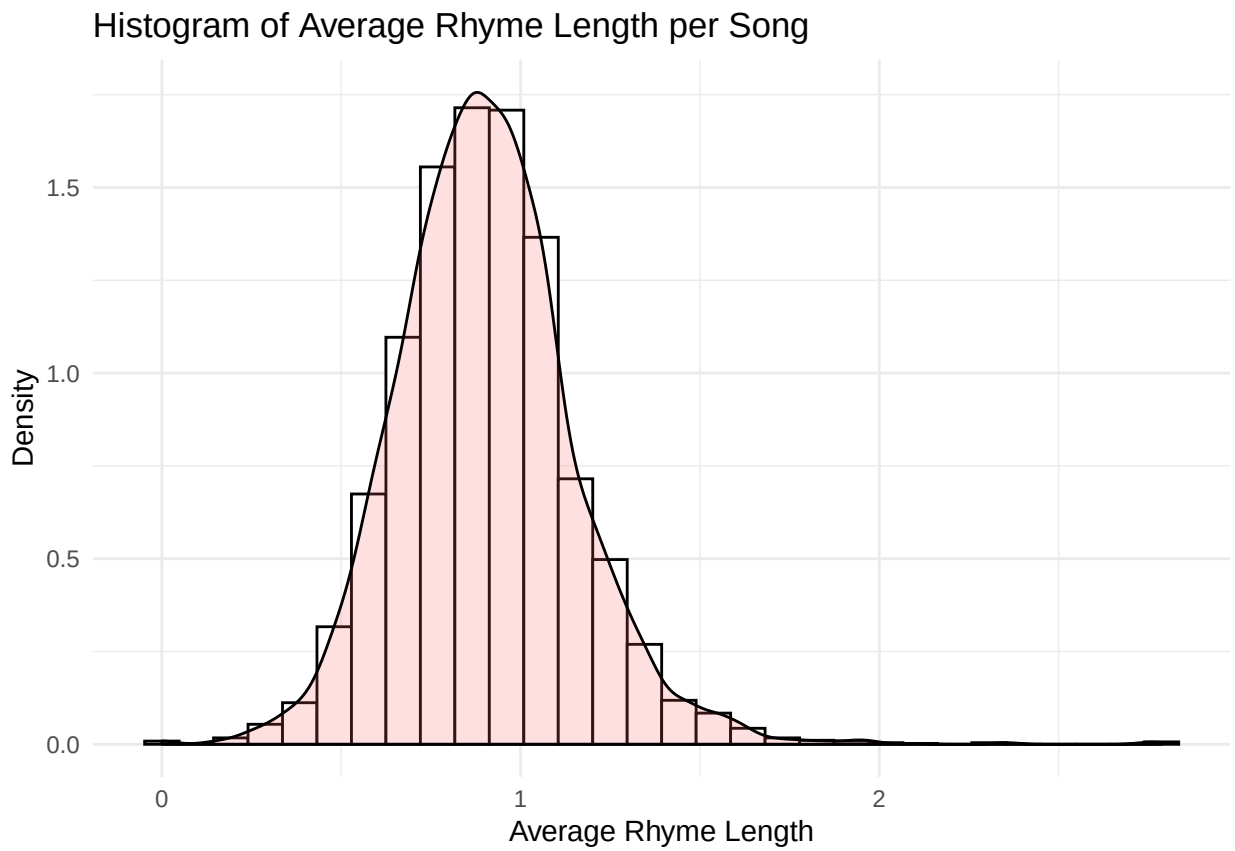
I'll plot the three most interesting of the X's that I mentioned above: Average Word Length, Average Rhyme Length, and Sentiment. Let's start with Average Word Length.

```
data = read.csv("data_clean.csv")
plot = ggplot(data,aes(x=AverageWordLength)) +
  geom_histogram(aes(y=..density..), colour="black", fill="white",bins=30) +
  geom_density(alpha=.2, fill="#FF6666") +
  labs(title = "Histogram of Average Word Length per Song",x="Number of Words",y="Density") +
  theme_minimal()
plot
```



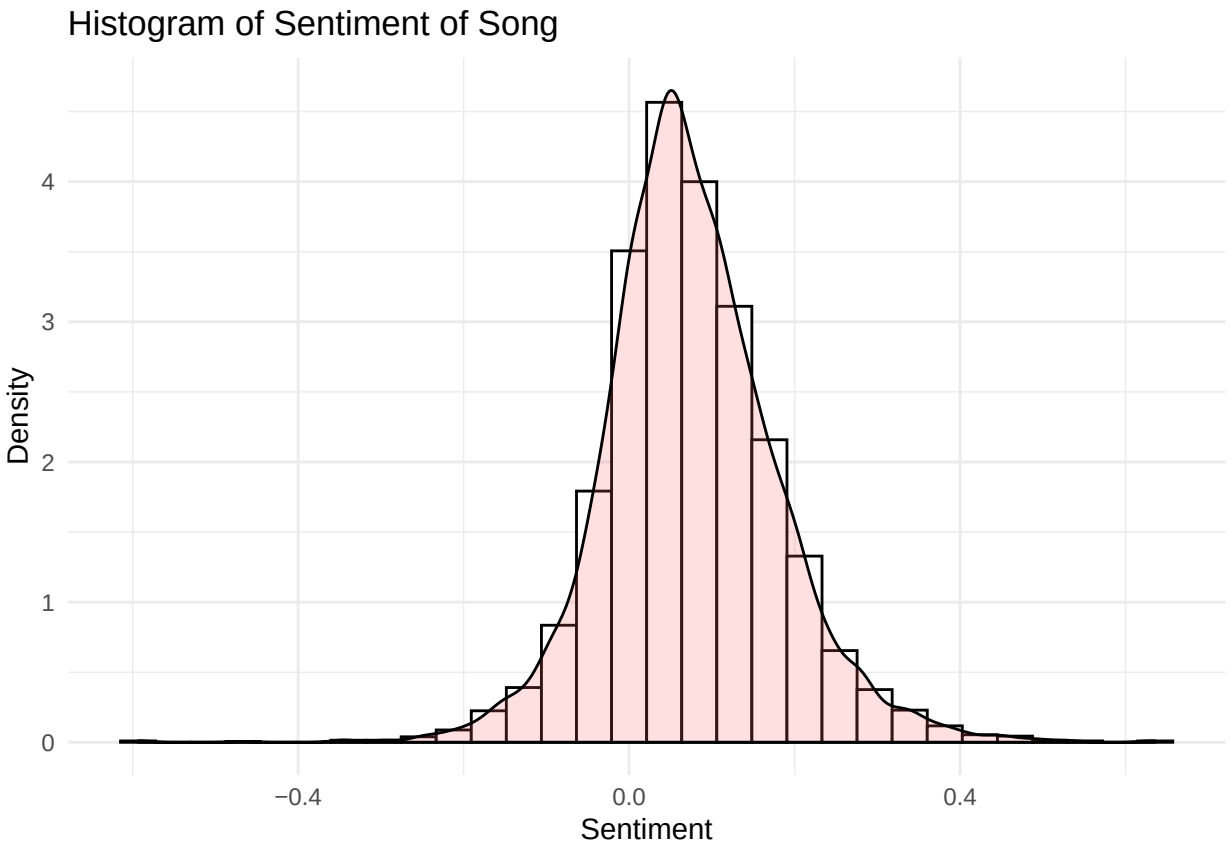
Next, Average Rhyrme Length.

```
plot = ggplot(data,aes(x=AverageRhymeLength)) +
  geom_histogram(aes(y=..density..), colour="black", fill="white",bins=30) +
  geom_density(alpha=.2, fill="#FF6666") +
  labs(title = "Histogram of Average Rhyme Length per Song",x="Average Rhyme Length",y="Density") +
  theme_minimal()
plot
```



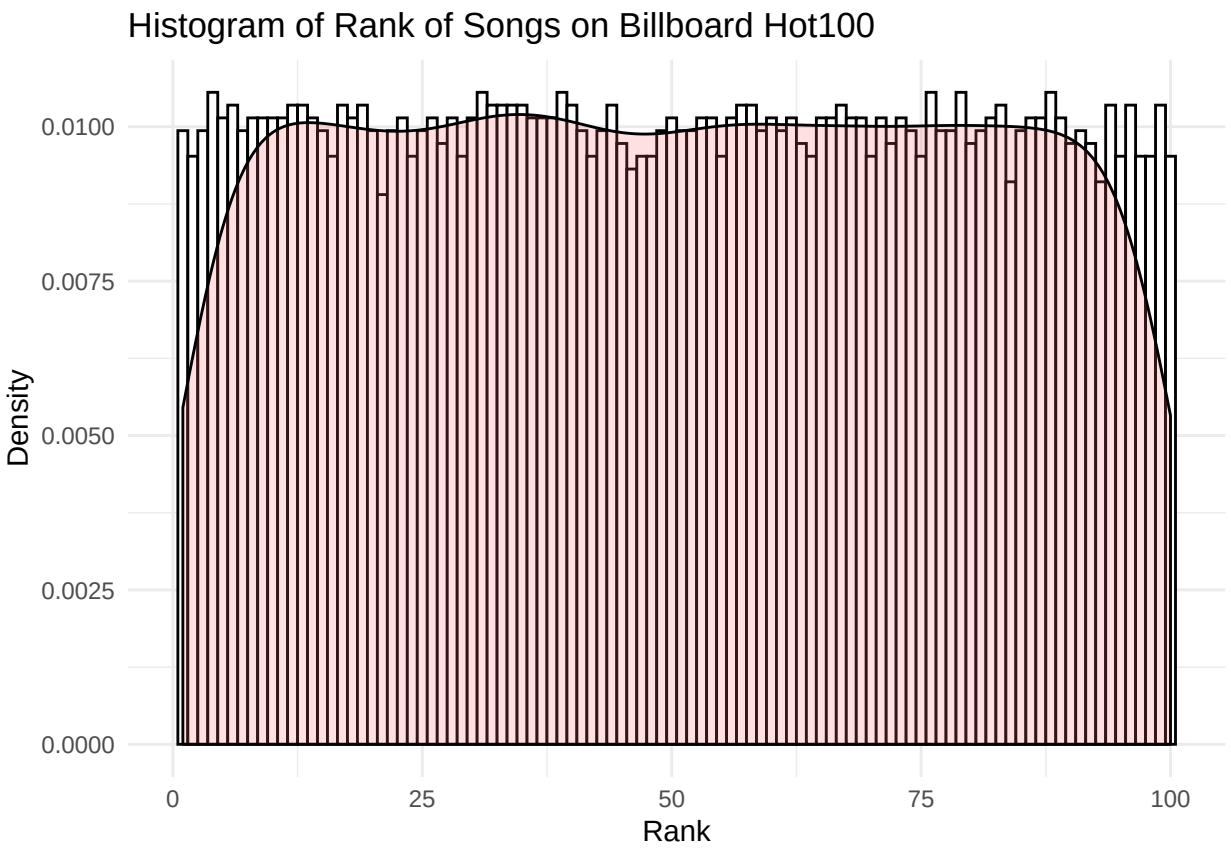
And Sentiment.

```
plot = ggplot(data,aes(x=Sentiment)) +
  geom_histogram(aes(y=..density..), colour="black", fill="white",bins=30) +
  geom_density(alpha=.2, fill="#FF6666") +
  labs(title = "Histogram of Sentiment of Song",x="Sentiment",y="Density") +
  theme_minimal()
plot
```



and finally, Rank on Billboard Hot100. This is technically a discrete variable, but it has 100 values so a histogram works better.

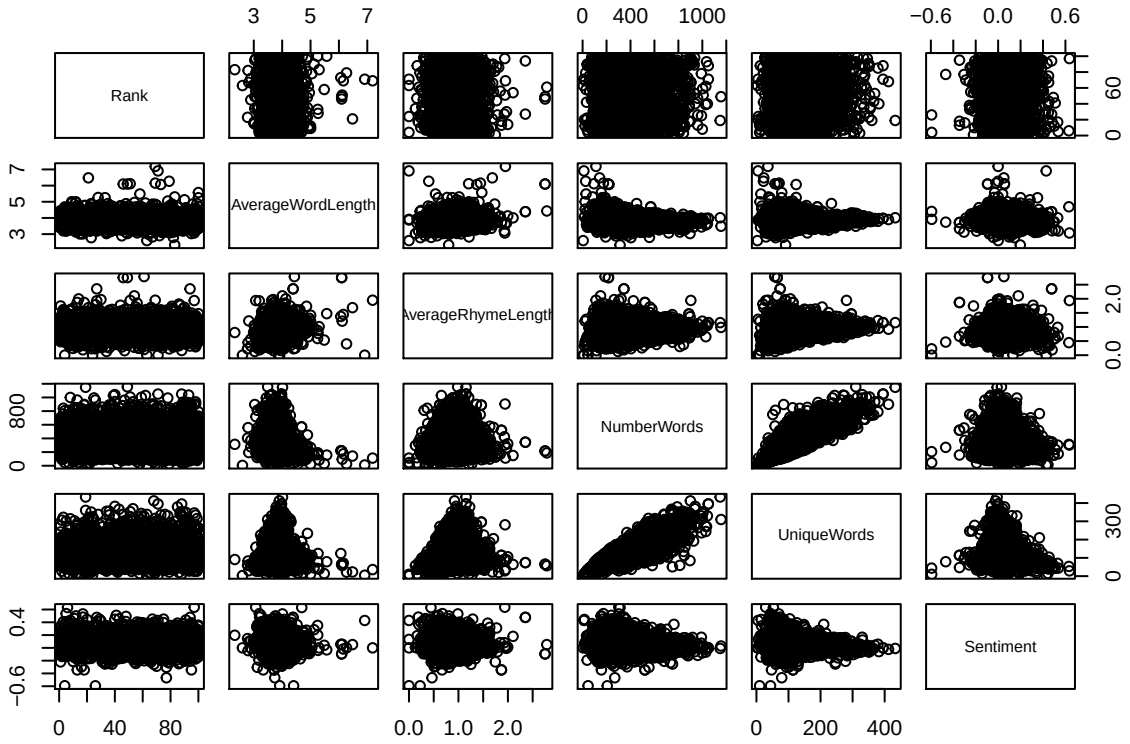
```
plot = ggplot(data,aes(x=Rank)) +
  geom_histogram(aes(y=..density..), colour="black", fill="white",bins=100) +
  geom_density(alpha=.2, fill="#FF6666") +
  labs(title = "Histogram of Rank of Songs on Billboard Hot100",x="Rank",y="Density") +
  theme_minimal()
plot
```




```
# I am curious why the histogram isn't totally uniform, as it should be. TODO: Find out why
```

And Correlation of Variables

```
d = data %>% select(.,Rank,AverageWordLength,AverageRhymeLength,NumberWords,UniqueWords,Sentiment)
pairs(d)
```


Rank
AverageWordLength
AverageRhymeLengt
NumberWords
UniqueWords
Sentiment