Predicting Song Popularity from Characteristics of Lyrics

Table of Contents (Clickable!)

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("../Report1/billboard_lyrics_1964-2015.csv")
names(original_data)
```

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

```
# head(original_data)
original_data[1:5,] %>% select(Rank,Song,Artist,Year) # Excluding Lyrics
```

```
##   Rank                                    Song
## 1    1                             wooly bully
## 2    2 i cant help myself sugar pie honey bunch
## 3    3               i cant get no satisfaction
## 4    4                     you were on my mind
## 5    5             youve lost that lovin feelin
##                         Artist Year
## 1 sam the sham and the pharaohs 1965
## 2                    four tops 1965
## 3           the rolling stones 1965
## 4                      we five 1965
## 5        the righteous brothers 1965
```

I decided that I wanted to come up with more columns to better interpret lyrics and had to come up with some independent 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've moved the code to this github if you want to see the new and updated python scripts.)

https://github.com/kevinterwilliger/Lyric-Analysis

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:

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 sequence 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 arbitrary 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. A negative score correlates to a more negative emotion, whether angry, sad, or both. Positive scores would work in the reverse, the higher the score, the more positive the sentiment expressed in the lyrics is.

Examining the Data

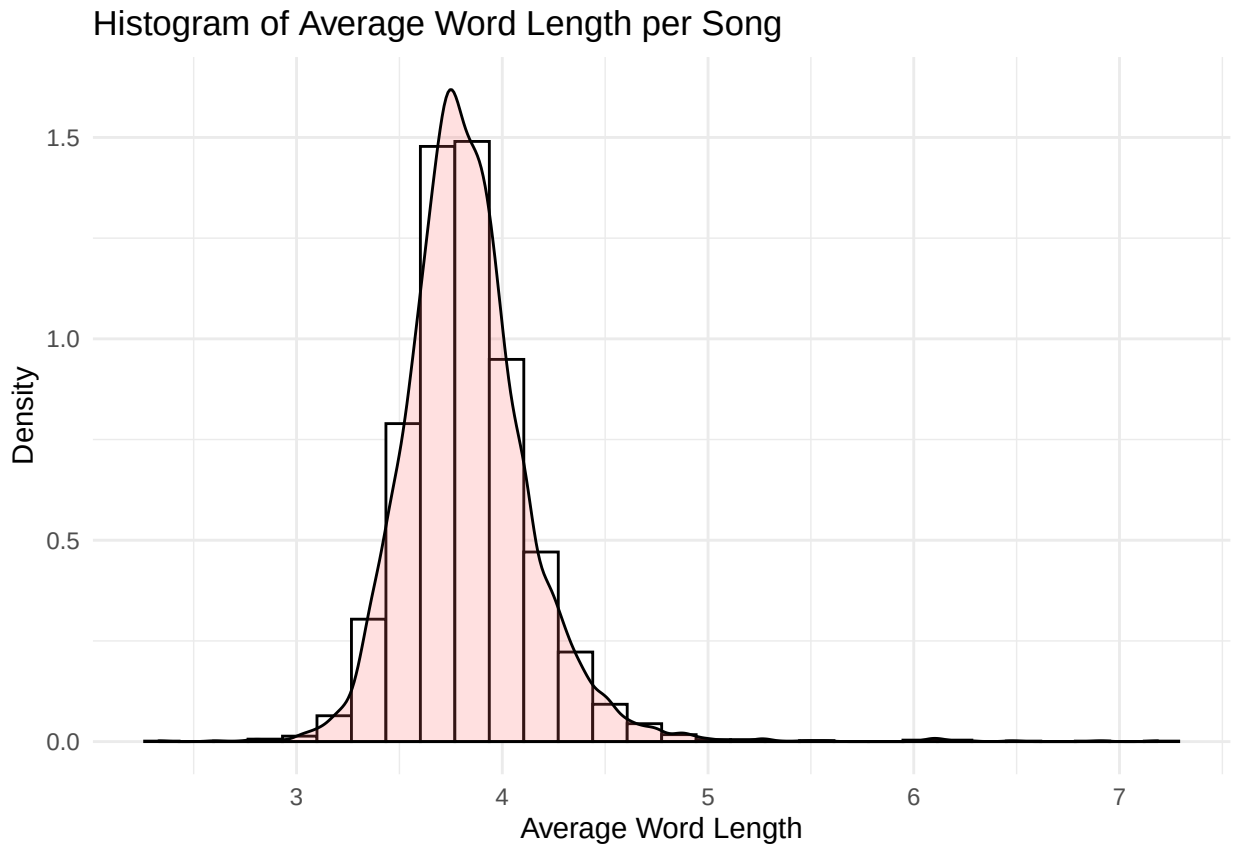
I'll plot the three most interesting of the X's that I mentioned above: Average Word Length, Average Rhyme Length, and Sentiment. Let's start with Average Word Length.

```
data2 = read.csv("data_clean.csv")
head(data2)
```

```
##   X.1 X Rank                                    Song
## 1   1 0    1                             wooly bully
## 2   2 1    2 i cant help myself sugar pie honey bunch
## 3   3 3    4                   you were on my mind
## 4   4 4    5          youve lost that lovin feelin
## 5   5 5    6                               downtown
## 6   6 6    7                                   help
##                         Artist Year
## 1 sam the sham and the pharaohs 1965
## 2                   four tops 1965
## 3                     we five 1965
## 4       the righteous brothers 1965
## 5                 petula clark 1965
## 6                  the beatles 1965
##
## 1
## 2
## 3
## 4
## 5  when youre alone and life is making you lonely you can always go downtown when youve got worries a
## 6
##   Source AverageWordLength AverageRhymeLength NumberWords UniqueWords
## 1      3          4.280000          1.7456140         125          64
## 2      1          3.872549          0.8484848         204          94
## 3      1          3.546053          0.7622378         152          44
## 4      1          4.051724          0.7982456         232          88
## 5      1          4.573222          0.6898148         239         120
## 6      3          3.780702          1.0772727         228          76
##     Sentiment
## 1 -0.21978022
## 2  0.13592233
```

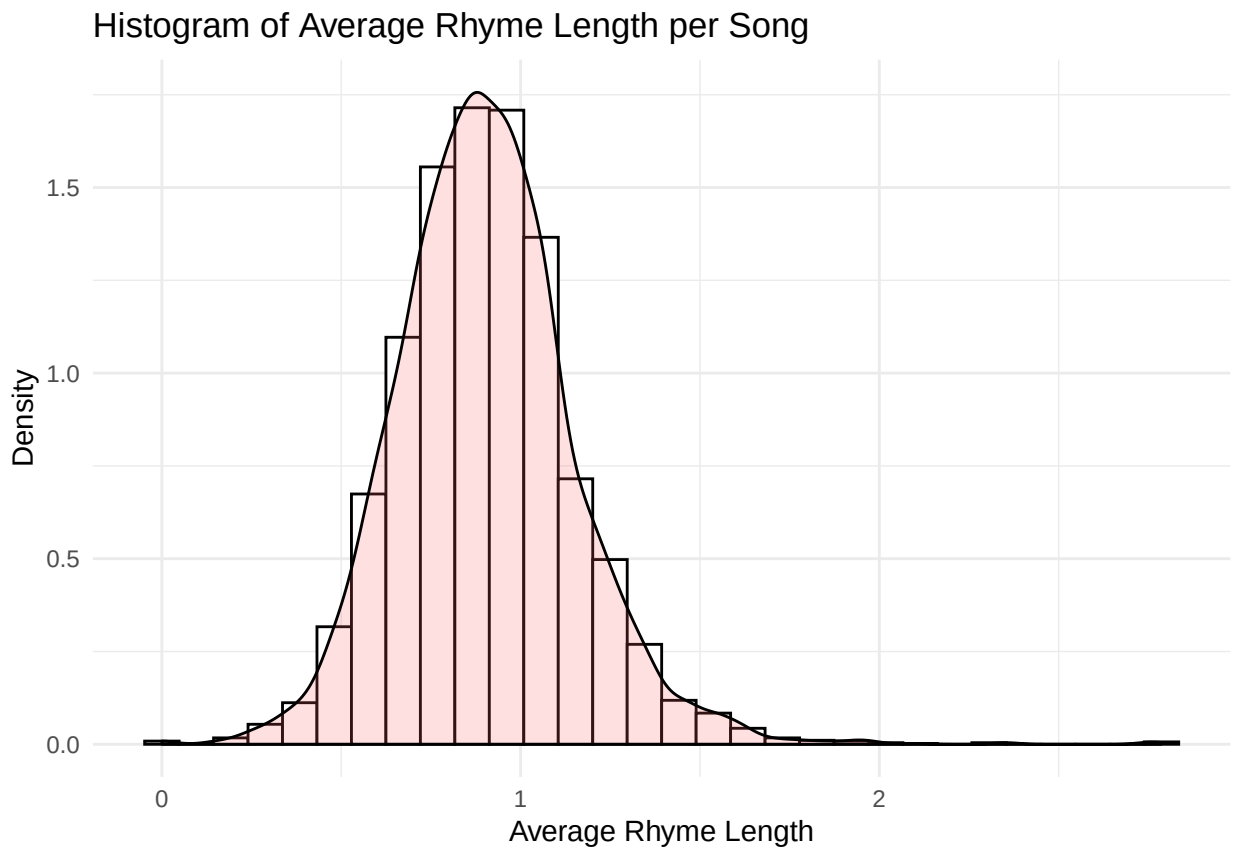
```
## 3 -0.12328767
## 4  0.10638298
## 5  0.01526718
## 6  0.24786325
```

```
plot = ggplot(data2,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="Average Word Length",y="Density") +
  theme_minimal()
plot
```



From the plot, we can see that the average word length in each song tends to be between 3 letters and 5 letters. I wonder how this histogram differs from the average word length of all english words. My best guess is that this graph is shifted left: not that many songs use words like supercalifragilisticexpialidocious. Next, Average Rhyme Length.

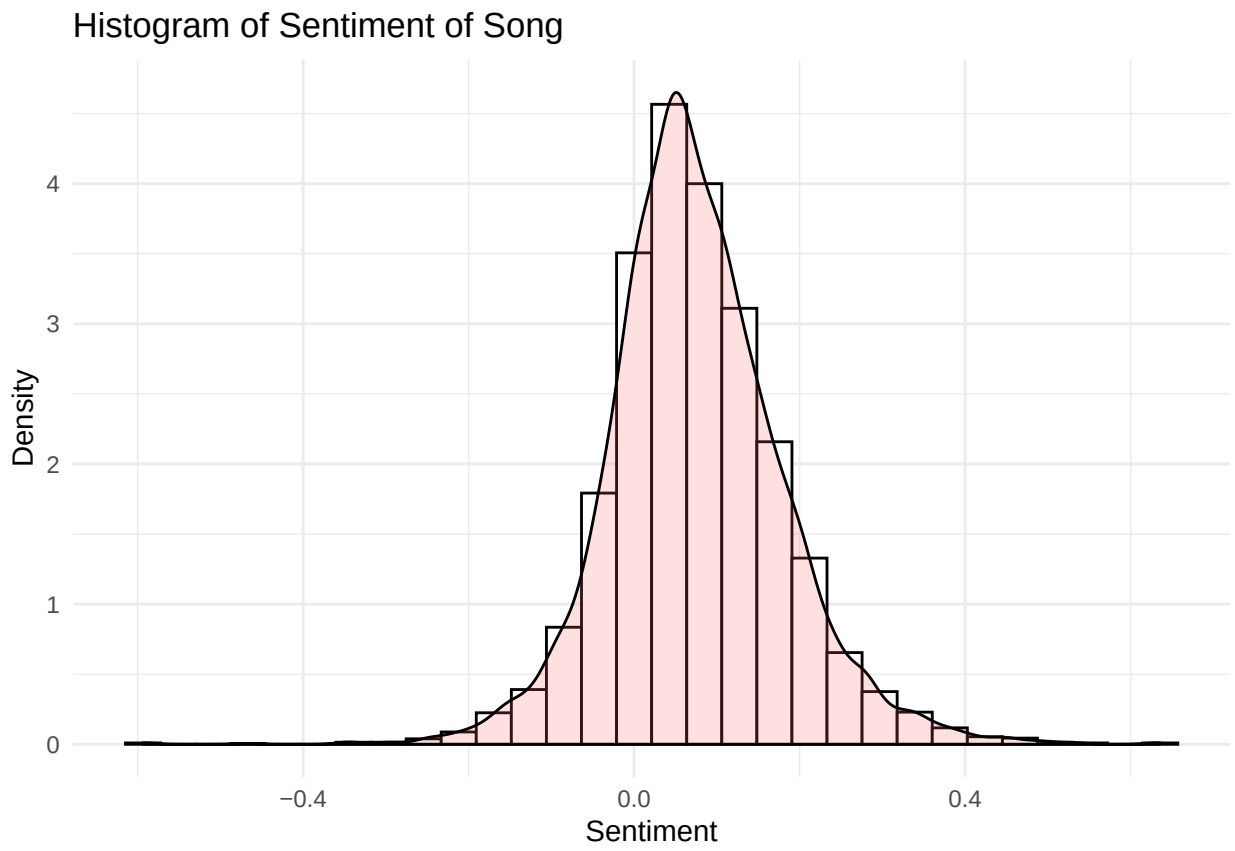
```
plot = ggplot(data2,aes(x=AverageRhymeLength)) +
  geom_histogram(aes(y=..density..), colour="black", fill="white",bins=30) +
  geom_density(alpha=.2, fill="#FF6666") +
  labs(title = "Histogram of Average Rhyme Length per Song",
       x="Average Rhyme Length",y="Density") +
  theme_minimal()

plot
```



So the average rhyme length in the songs is around 1 vowel-phonetic per rhyme. I am curious to see how this might correlate to genre (as a hip-hop fan, I hope that rap might have longer rhymes than pop or country).

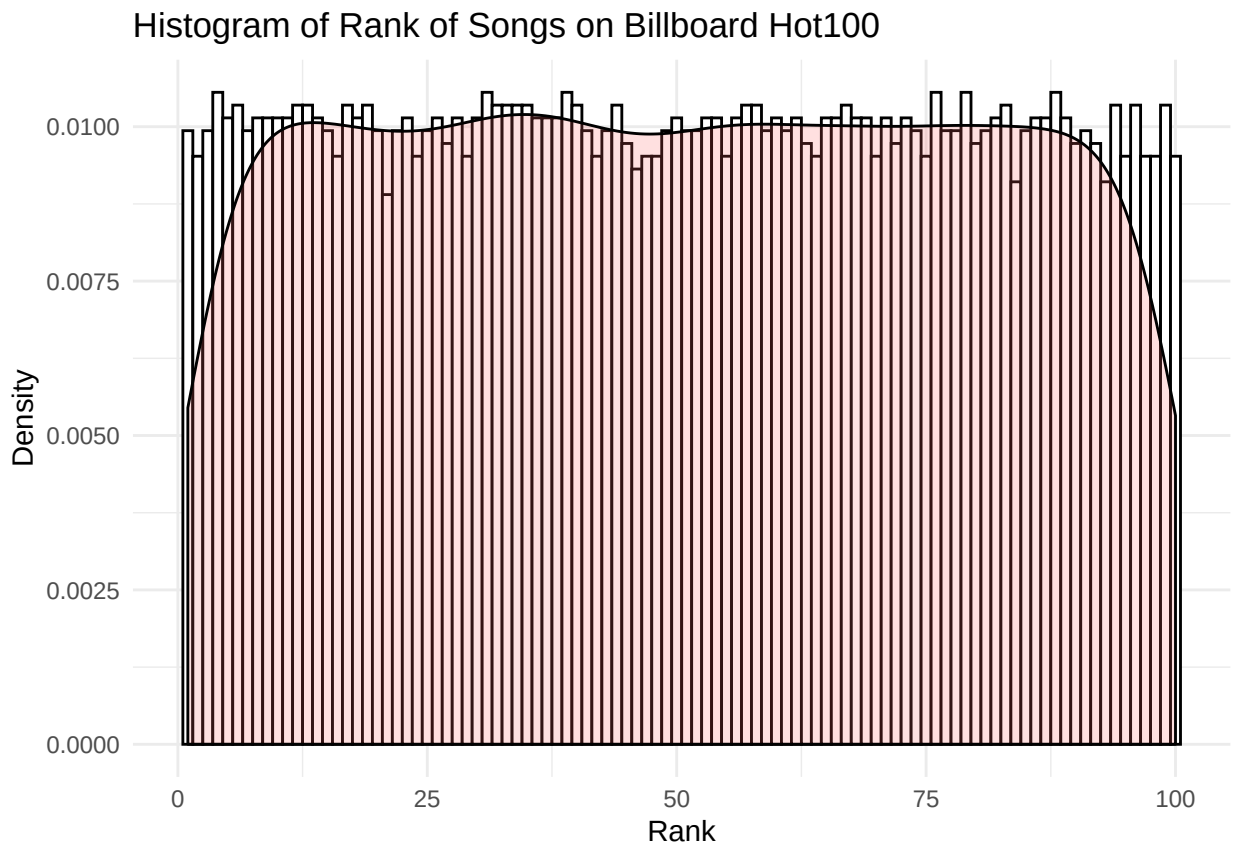
And Sentiment.

```
plot = ggplot(data2,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
```



So most songs tend to skew positive, I guess that's not terribly surprising. And finally, Rank on Billboard Hot100. This is technically a discrete variable, but it has 100 values so a histogram works better.

```
plot = ggplot(data2,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
```

The question above actually has two answers. The first is because this "data" is cleaned to remove any junk values I appended in the script, therefore having less that 50*100 rows and a non-uniform histogram. The other answer I will show below.

```
# get the count of every song/artist combination
proof = data2 %>%
  mutate(track = paste(Song,Artist,sep=" ")) %>%
  group_by(track) %>%
  summarise(n = n()) %>%
  filter(n > 1)
# Checking for duplicates
proof
```

```
## Registered S3 method overwritten by 'cli':
##   method     from
##   print.tree tree
```

```
## # A tibble: 195 x 2
##    track                                  n
##    <chr>                              <int>
##  1 100 pure love crystal waters           2
##  2 3 britney spears                       2
##  3 4 seasons of loneliness boyz ii men    2
##  4 adorn miguel                           2
##  5 again janet jackson                    2
##  6 all about that bass meghan trainor     2
```

```
##  7 all cried out allure featuring 112      2
##  8 all for you sister hazel                2
##  9 all i wanna do sheryl crow              2
## 10 all that she wants ace of base          2
## # ... with 185 more rows
```

So there are 195 duplicates. Out of curiosity, are these duplicates an error by whoever created the dataset?

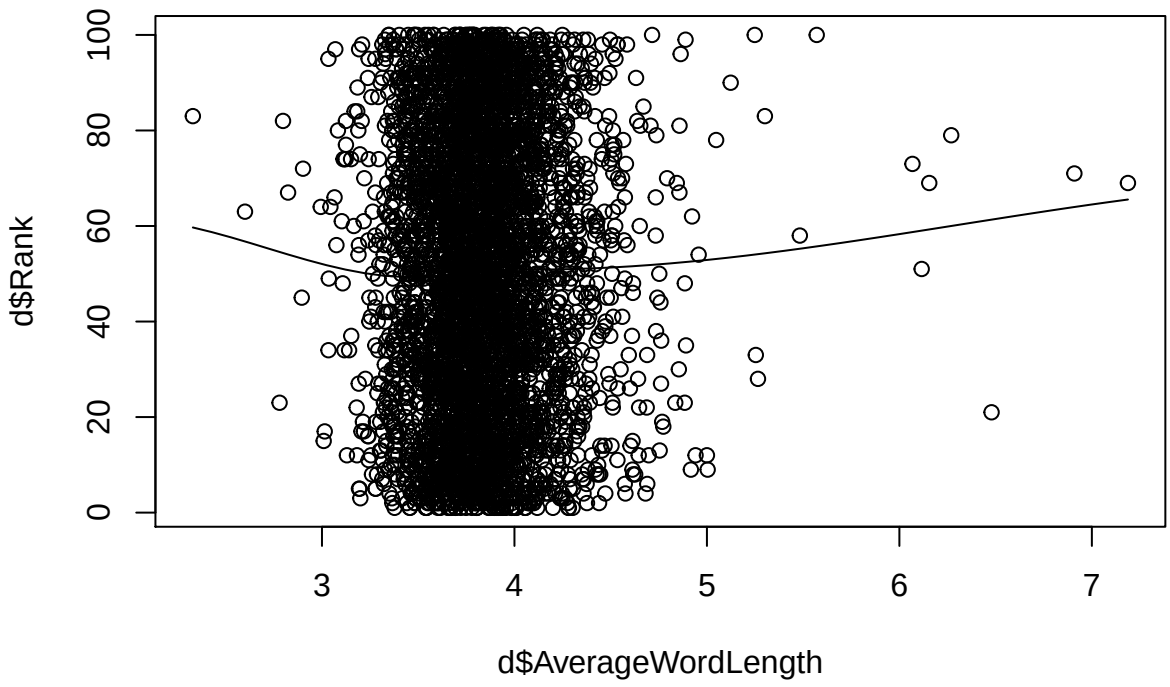
```
data2$Year[data2$Song == "100 pure love"]
```

```
## [1] 1994 1995
```

I'm going to say that the creator did nothing wrong and that the vast majority, if not all, duplicates are caused by songs being popular for more than just one year. I think I will remove duplicates in the future.

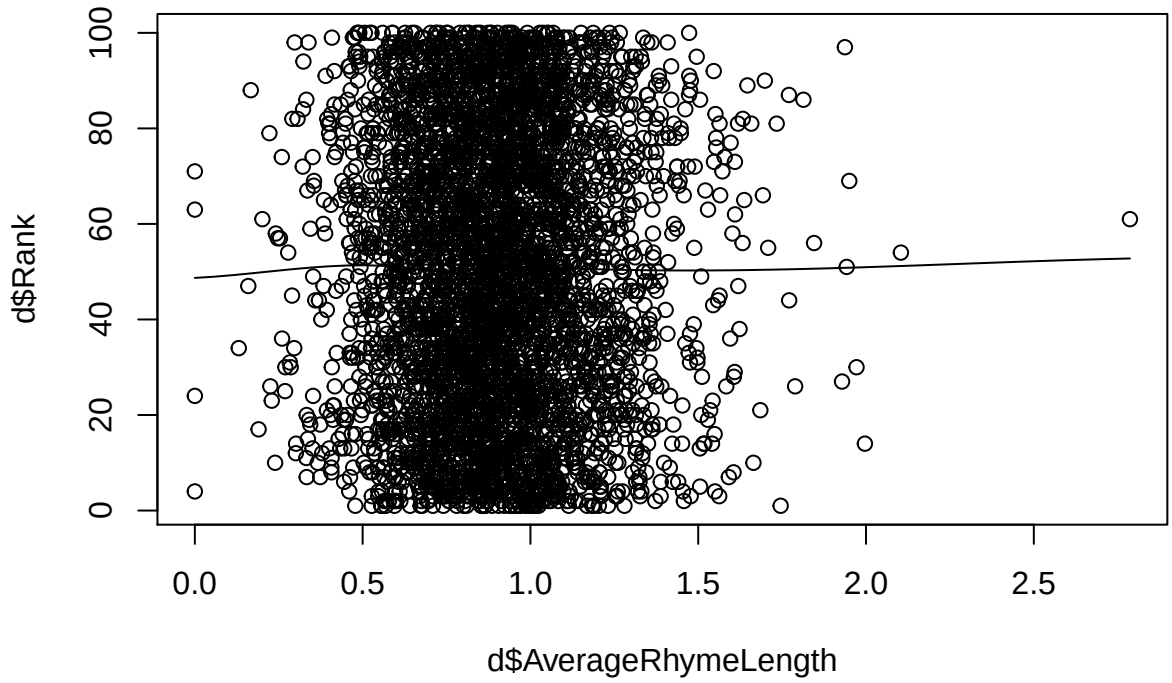
Correlation of Variables

```
d = data2 %>% dplyr::select(Rank,AverageWordLength,AverageRhymeLength,NumberWords,UniqueWords,Sentiment
scatter.smooth(d$Rank ~ d$AverageWordLength,data=d)
```



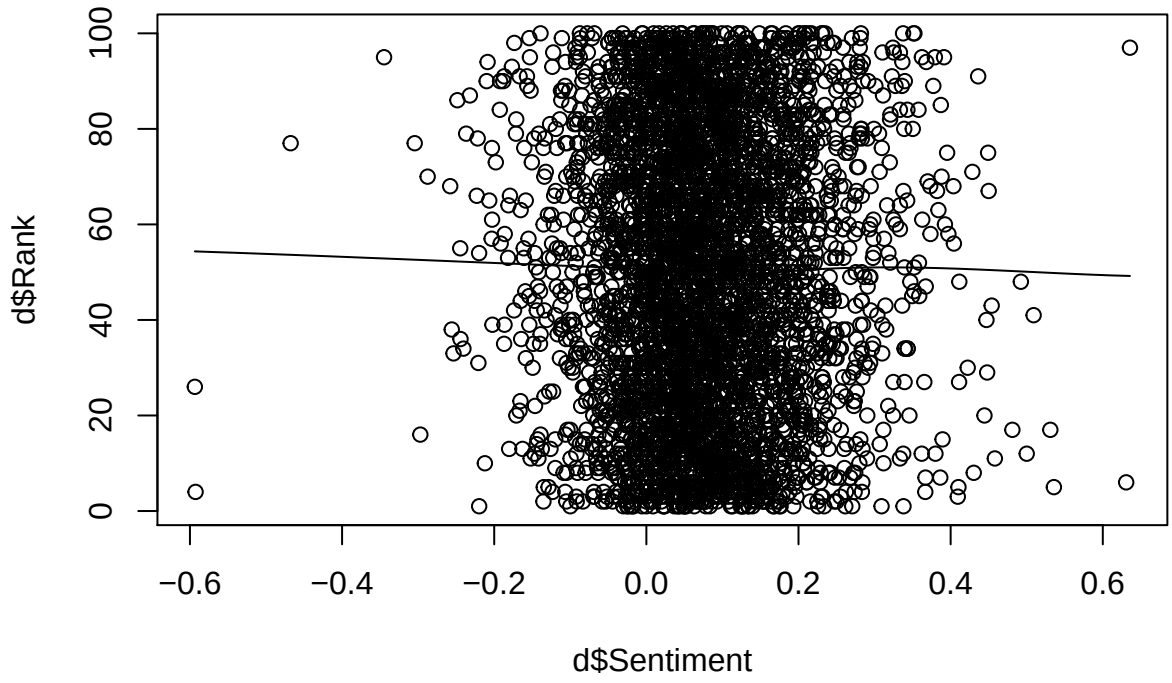
Not a high correlation.

```
scatter.smooth(d$Rank ~ d$AverageRhymeLength)
```



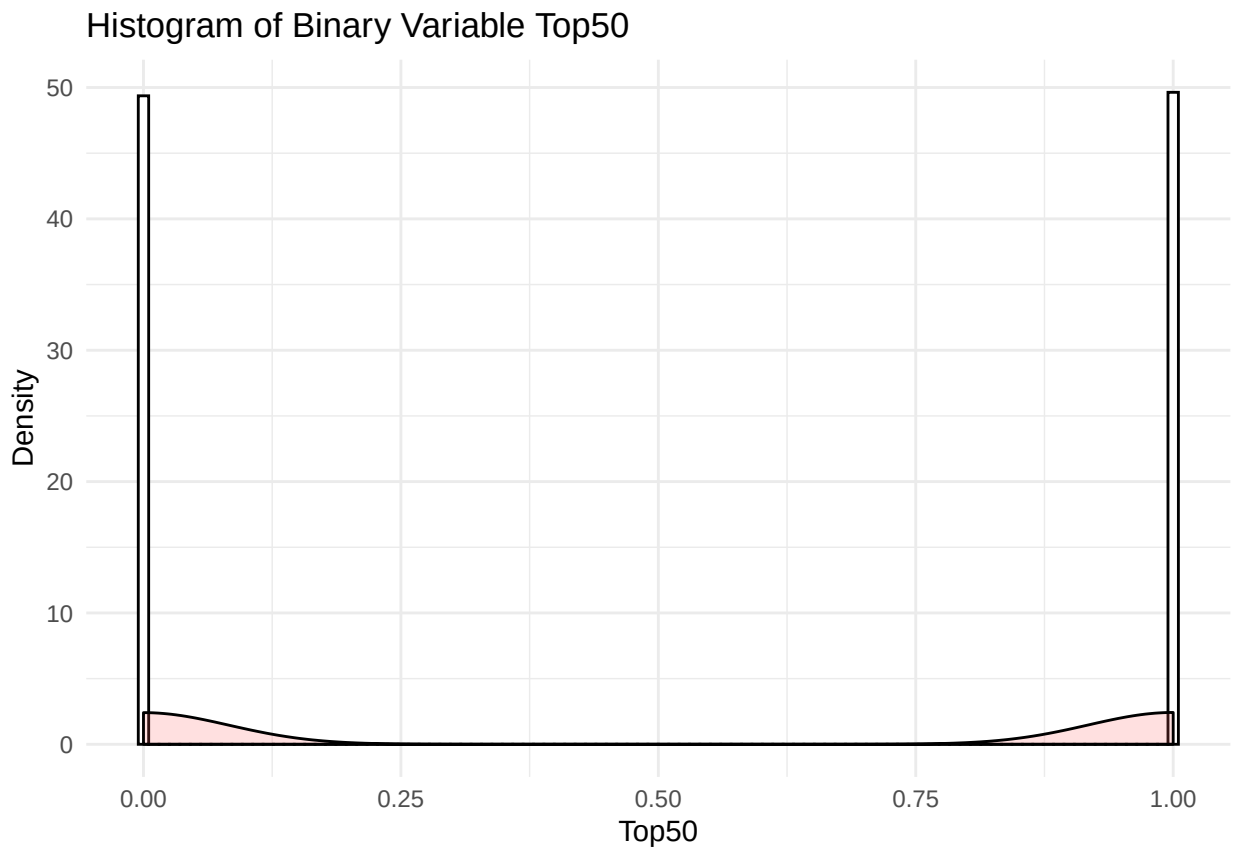
Agian, low.

```
scatter.smooth(d$Rank ~ d$Sentiment)
```

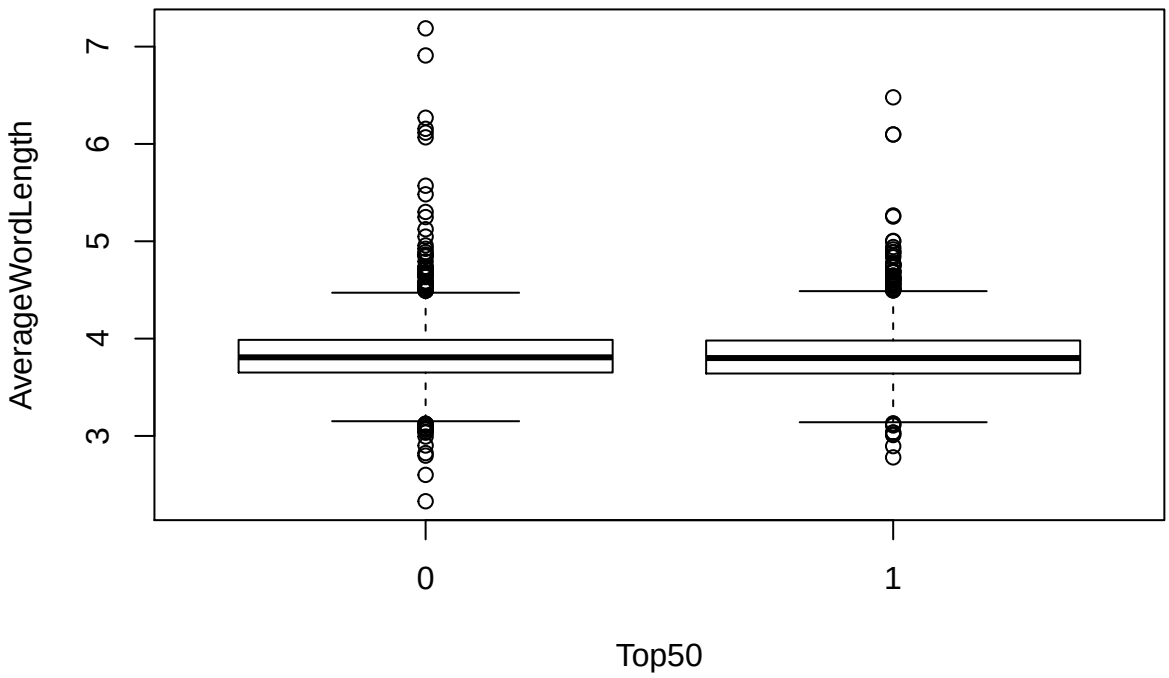


Yeah, so not good. I can't say I'm surprised, but there has got to be a way for me to get better predictions from these variables. First, maybe a binary prediction variable will prove better.

```
binary = data_ %>%
  mutate(Top50 = ifelse(Rank <= 50,1,0)) %>%
  select(Top50,AverageWordLength,AverageRhymeLength,Sentiment,NumberWords,UniqueWords)

plot = ggplot(binary,aes(x=Top50)) +
  geom_histogram(aes(y=..density..), colour="black", fill="white",bins=100) +
  geom_density(alpha=.2, fill="#FF6666") +
  labs(title = "Histogram of Binary Variable Top50",x="Top50",y="Density") +
  theme_minimal()
plot
```

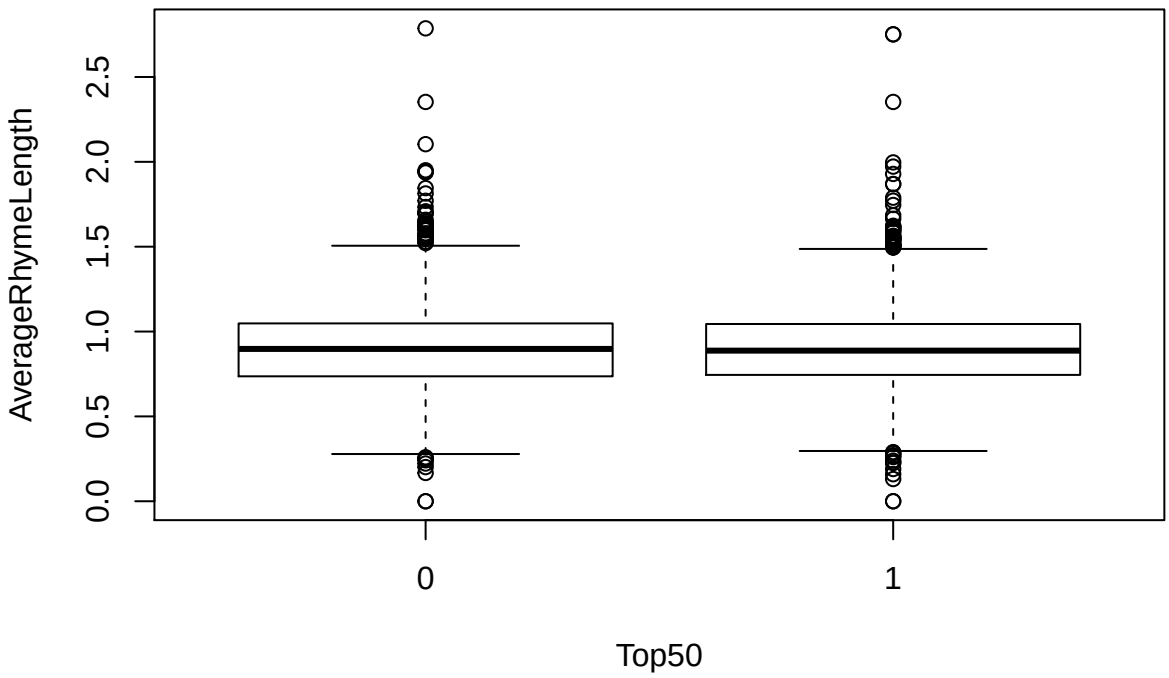


OK good, a 50-50(ish) split is what we wanted.



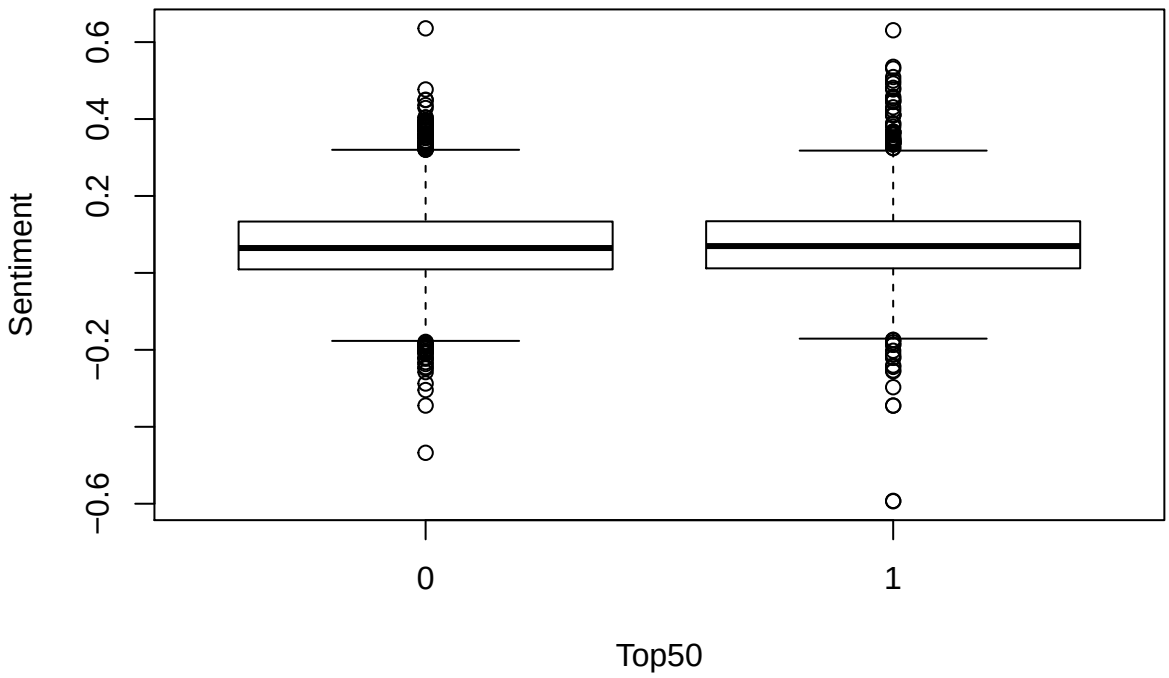
Oof. Not good.

```
boxplot(AverageRhymeLength ~ Top50,data=binary)
```



This is not an improvement.

```
boxplot(Sentiment ~ Top50,data=binary)
```



Yeah so not better. In the next report, I will be adding a few more columns, one of them being genre of the song. I will probably try to predict genre from the other columns, hopefully it will work better.

Concluding Thoughts

I really want to make this project work. So far it seems likely that the linear regressions and logistic regressions in report 2 will prove to be inept at predicting Rank or Top50. As I write this conclusion, I am trying to finalize the script I mentioned above, but it relys on some APIs that are proving to be difficult. I think that trying to predict genre from variables like average rhyme length or unique words might prove more successful than predicting popularity. If the graders have any recommendations on possible Y/X variables, I'd very much appreciate the outside input. Other than that, here is report 2!

Report 2

Introduction

Due to the fact that any linear regressions trying to predict the popularity of a song from the aforementioned independent variables are very underwhelming, I wrote some more python scripts (that can be found at https://github.com/kevinterwilliger/Lyric-Analysis) to collect more variables such as the key or mode of a song and the genres of a song. I used the Spotipy library to collect an audio analysis of the song that contains the following variables:

1. **Loudness** - The overall loudness (decibels, dB) of a song.
2. **Tempo** - The beats per minute of the song.
3. **Key** - The estimated overall key of the song. Ranges in values from 0-11 mapping to pitches, using the standard Pitch Class Notation.

4. **Mode** - Binary variable where "0" is Minor and "1" is in Major
5. **Time Signiture** - The "meter" of a song is measured by the number beats in each bar. Values range from 3-7 and indicate "$\frac{3}{4}$"-"$\frac{7}{4}$".

Using the Last.FM API, I was able to collect the top 5 user tags corresponding to each song. I collected 5 tags due to the fact that tags do not always contain the genre, but most songs have the specific genre in one of the 5 tags. I will explain my process for narrowing the tags down further in the report.

Regressions

Rank

Let's start with running some regressions on Rank. The results are not great, but I expected this due to the fact popularity probably is not dictated by the contents of the lyrics. We'll touch on Genre after.

Average Word Length

```
fit = lm(Rank ~ AverageWordLength, data=data)
summary(fit)
```

```
##
## Call:
## lm(formula = Rank ~ AverageWordLength, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -50.010 -24.910  -0.068  25.223  50.346
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)         44.904      5.407   8.305   <2e-16 ***
## AverageWordLength    1.419      1.406   1.009    0.313
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.93 on 4439 degrees of freedom
## Multiple R-squared:  0.0002294,  Adjusted R-squared:  4.156e-06
## F-statistic: 1.018 on 1 and 4439 DF,  p-value: 0.3129
```

From the above summary, the R-squared of the linear regression of average word length on rank is 0.00005306. This is such a bad R-squared that it is clear that average word length has almost no bearing on how popular a song will be.

Average Rhyme Length

```
fit = lm(Rank ~ AverageRhymeLength,data=data)
summary(fit)
```

```
##
## Call:
## lm(formula = Rank ~ AverageRhymeLength, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -49.774 -25.188  -0.265  25.459  49.867
##
```

```
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)          49.8867     1.6550  30.143   <2e-16 ***
## AverageRhymeLength   0.5081     1.7795   0.286    0.775
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.93 on 4439 degrees of freedom
## Multiple R-squared:  1.837e-05,  Adjusted R-squared:  -0.0002069
## F-statistic: 0.08153 on 1 and 4439 DF,  p-value: 0.7753
```

Again, a terrible R-squared. It is interesting, though, that average rhyme length has a tiny negative relationship toward rank.

Number of Words

```
fit = lm(Rank ~ NumberWords, data=data)
summary(fit)
```

```
##
## Call:
## lm(formula = Rank ~ NumberWords, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -49.944 -24.872  -0.063  25.188  51.058
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 51.319905   0.950487  53.993   <2e-16 ***
## NumberWords -0.003007   0.002602  -1.156    0.248
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.93 on 4439 degrees of freedom
## Multiple R-squared:  0.0003008,  Adjusted R-squared:  7.556e-05
## F-statistic: 1.336 on 1 and 4439 DF,  p-value: 0.2479
```

I would expect a small or negative relationship from this due to the fact that long songs do not often recieve as many listens as shorter songs.

Number of Unique Words

```
fit = lm(Rank ~ UniqueWords, data = data)
summary(fit)
```

```
##
## Call:
## lm(formula = Rank ~ UniqueWords, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -51.262 -24.871  -0.055  25.226  50.480
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept) 49.027572   1.006923  48.690   <2e-16 ***
## UniqueWords  0.011720   0.008097   1.447    0.148
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.92 on 4439 degrees of freedom
## Multiple R-squared:  0.0004718,  Adjusted R-squared:  0.0002466
## F-statistic: 2.095 on 1 and 4439 DF,  p-value: 0.1478
```

This is the most significant variable, it being able to pass a 85% significance test. It is a shame, though, that the R-squared is basically zero.

Sentiment

```
fit = lm(Rank~Sentiment,data=data)
summary(fit)
```

```
##
## Call:
## lm(formula = Rank ~ Sentiment, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -50.117 -25.018  -0.074  25.233  50.383
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  50.5407     0.5335  94.730   <2e-16 ***
## Sentiment    -2.6221     4.1079  -0.638    0.523
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.93 on 4439 degrees of freedom
## Multiple R-squared:  9.178e-05,  Adjusted R-squared:  -0.0001335
## F-statistic: 0.4074 on 1 and 4439 DF,  p-value: 0.5233
```

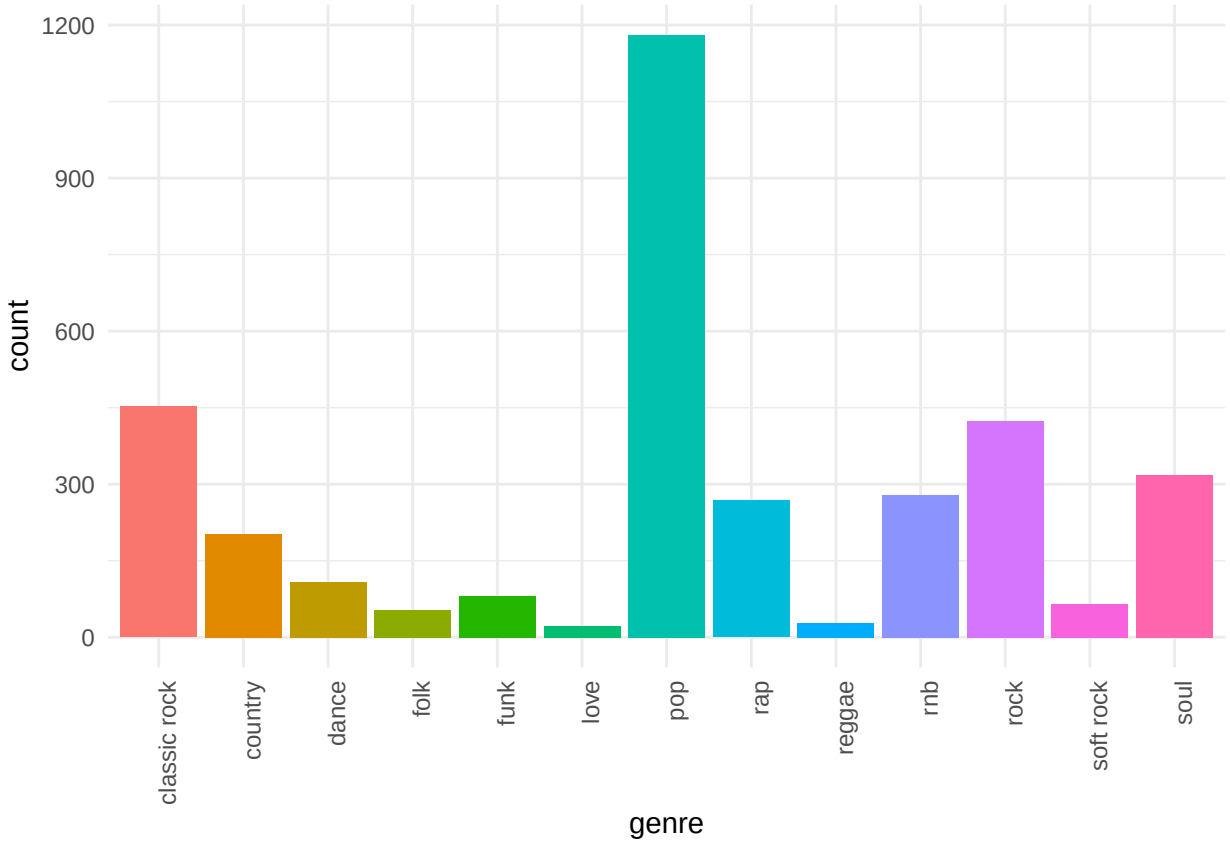
Since none of my lyrics columns are good predictors of Rank, I think that using a classification model to predict genres from lyrics might prove more viable than a regression on rank.

Genre

Next, let's use genre as the variable we are trying to predict.

Due to the nature of the API, I ended up dropping around 187 songs, but that leaves us with 3931 rows of song data, genre included. Lets look at the spread of the genres.

```
num_genre = clean_full %>% group_by(genre) %>% summarise(count=n()) %>% select(genre,count)

genreplot = num_genre[num_genre$count > 20,] %>%
  ggplot(., aes(x=genre, y=count, fill=genre)) +
  geom_bar(stat="identity")+theme_minimal()+
  theme(axis.text.x = element_text(angle = 90, hjust = 1),
        legend.position = 'none')

genreplot
```



There are about 1200 pop songs, but pop is such a wide ranging genre that this should be somewhat expected. Other than pop, it seems that these 17 genres are pretty all-representing.

This cleans the genres with under 20 songs and creates binary columns representing the 17 genres.

```
genre_list = num_genre[num_genre$count > 20,] %>% mutate(X = row_number())

clean_cut = clean_full[clean_full$genre %in% genre_list$genre,] %>%
  mutate(X = row_number())


binary = as.data.frame(clean_cut %>% select(X,genre) %>% model.matrix(~.,data=.))

final_data = right_join(clean_cut,binary,by="X",keep=F)
colnames(final_data)
```

```
##  [1] "X.2"              "X.1"              "Unnamed..0"
##  [4] "X"                "Rank"             "Song"
##  [7] "Artist"           "Year"             "Lyrics"
## [10] "Source"           "AverageWordLength" "AverageRhymeLength"
## [13] "NumberWords"      "UniqueWords"      "Sentiment"
## [16] "Loudness"         "Tempo"            "Key"
## [19] "Mode"             "time_signature"   "feature"
## [22] "genre"            "track"            "(Intercept)"
## [25] "genrecountry"     "genredance"       "genrefolk"
## [28] "genrefunk"        "genrelove"        "genrepop"
## [31] "genrerap"         "genrereggae"      "genrernb"
```

```
## [34] "genrerock"          "genresoft rock"     "genresoul"
```

```
head(final_data$Loudness)
```

```
## [1] -11.769 -21.836 -16.492 -11.763 -12.992 -26.048
```

Now that we have our genre columns, we can try to predict the genre from the different independent variables.

Predicting Pop songs

First lets try just the lyric data.

```
pop = final_data %>% select(genrepop,
                            Rank,
                            AverageWordLength,
                            AverageRhymeLength,
                            NumberWords,
                            UniqueWords,
                            Sentiment)

fit = glm(genrepop ~ Rank +
            AverageWordLength +
            AverageRhymeLength +
            NumberWords +
            UniqueWords +
            Sentiment,data=pop,family = "binomial")

summary(fit)
```

```
##
## Call:
## glm(formula = genrepop ~ Rank + AverageWordLength + AverageRhymeLength +
##     NumberWords + UniqueWords + Sentiment, family = "binomial",
##     data = pop)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4538  -0.9439  -0.7841   1.3365   2.2356
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.0754469  0.5206643  -0.145 0.884786
## Rank               -0.0074870  0.0012676  -5.907 3.49e-09 ***
## AverageWordLength   0.0972550  0.1289539   0.754 0.450739
## AverageRhymeLength  0.1621574  0.1500988   1.080 0.279992
## NumberWords         0.0016199  0.0004381   3.698 0.000217 ***
## UniqueWords        -0.0117302  0.0014862  -7.893 2.96e-15 ***
## Sentiment          -0.0254475  0.3454554  -0.074 0.941278
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4459.5  on 3480  degrees of freedom
## Residual deviance: 4325.9  on 3474  degrees of freedom
## AIC: 4339.9
```

```
##
## Number of Fisher Scoring iterations: 4
```

From this it seems like Number of Words, Unique Words, and Rank are all significant. Let's drop the insignificant variables.

```
pop = pop %>% select(Rank,genrepop,NumberWords,UniqueWords)

fit = glm(genrepop ~ Rank + NumberWords + UniqueWords,data=pop,family=binomial)
summary(fit)
```

```
##
## Call:
## glm(formula = genrepop ~ Rank + NumberWords + UniqueWords, family = binomial,
##     data = pop)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.4130  -0.9461  -0.7843   1.3358   2.2175
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.4355607  0.1172066   3.716 0.000202 ***
## Rank        -0.0075186  0.0012669  -5.935 2.94e-09 ***
## NumberWords  0.0014610  0.0004119   3.547 0.000390 ***
## UniqueWords -0.0111974  0.0014109  -7.936 2.09e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4459.5  on 3480  degrees of freedom
## Residual deviance: 4327.8  on 3477  degrees of freedom
## AIC: 4335.8
##
## Number of Fisher Scoring iterations: 4
```

And let's then create a confusion matrix.

```
probs = predict(fit,type="response")
prediction = predict(fit)
prediction = rep("Not Pop",nrow(pop))
prediction[probs > .5]="Pop"
real = recode(pop$genrepop,"1"="Pop","0"="Not Pop")
table(prediction,real)
```

```
##           real
## prediction Not Pop  Pop
##    Not Pop    2245 1160
##    Pop          55   21
```

The confusion matrix tells us that the model predicts that a song is not pop very accuratly, but seems unable to predict when a song is pop. Let's see if this is the case for other genres too.

Predicting Rock Songs

Using the same logic from above, here is the fit summary for predicting whether a song is rock or not.

```
rock = final_data %>% select(genrerock,
                             Rank,
                             AverageWordLength,
                             AverageRhymeLength,
                             NumberWords,
                             UniqueWords,
                             Sentiment)

fit = glm(genrerock ~ Rank +
            AverageWordLength +
            AverageRhymeLength +
            NumberWords +
            UniqueWords +
            Sentiment,data=rock,family = "binomial")

summary(fit)
```

```
##
## Call:
## glm(formula = genrerock ~ Rank + AverageWordLength + AverageRhymeLength +
##     NumberWords + UniqueWords + Sentiment, family = "binomial",
##     data = rock)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9798  -0.5598  -0.4763  -0.3716   2.5276
##
## Coefficients:
##                       Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -2.9967693  0.7120522  -4.209 2.57e-05 ***
## Rank                0.0053326  0.0018302   2.914 0.003571 **
## AverageWordLength   0.3888353  0.1738979   2.236 0.025352 *
## AverageRhymeLength  0.1827987  0.2137877   0.855 0.392525
## NumberWords        -0.0026835  0.0007184  -3.735 0.000188 ***
## UniqueWords        -0.0001691  0.0021890  -0.077 0.938413
## Sentiment          -1.6573110  0.5038994  -3.289 0.001006 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2579.4  on 3480  degrees of freedom
## Residual deviance: 2498.9  on 3474  degrees of freedom
## AIC: 2512.9
##
## Number of Fisher Scoring iterations: 5
```

Interestingly, there are different significant variables. Rank and Number of Words are the same, but this time sentiment is significant. What is interesting is that the coefficient for sentiment is much larger than any other variable from rock or pop.

```
rock = rock %>% select(Rank,genrerock,NumberWords,Sentiment)

fit = glm(genrerock ~ Rank + NumberWords + Sentiment,data=rock,family=binomial)
summary(fit)
```

```
##
## Call:
## glm(formula = genrerock ~ Rank + NumberWords + Sentiment, family = binomial,
##     data = rock)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9267  -0.5587  -0.4820  -0.3770   2.4828
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.2754856  0.1677819  -7.602 2.91e-14 ***
## Rank         0.0052733  0.0018224   2.894 0.003810 **
## NumberWords -0.0028971  0.0004252  -6.814 9.51e-12 ***
## Sentiment   -1.7587670  0.4975411  -3.535 0.000408 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2579.4  on 3480  degrees of freedom
## Residual deviance: 2505.3  on 3477  degrees of freedom
## AIC: 2513.3
##
## Number of Fisher Scoring iterations: 5
```

And let's then create a confusion matrix.

```
probs = predict(fit,type="response")
prediction = predict(fit)
prediction = rep("Not Rock",nrow(rock))
prediction[probs > .5]="Rock"
real = recode(rock$genrerock,"1"="Rock","0"="Not Rock")
table(prediction,real)
```

```
##           real
## prediction Not Rock Rock
##   Not Rock     3057  424
```

```
unique(prediction)
```

```
## [1] "Not Rock"
```

The last function I call is to prove that the model only predicts "Not Rock" (0) for any input in the data set. Due to that, the model can't be a good predictor if it can't predict that a rock song is a rock song. I'm going to remove pop songs from the data set to see if it helps.

```
notPop = final_data[final_data$genrepop == 0,]

rock = notPop %>% select(genrerock,
                         Rank,
                         AverageWordLength,
                         AverageRhymeLength,
                         NumberWords,
                         UniqueWords,
```

```
                          Sentiment)

fit = glm(genrerock ~ Rank + NumberWords + Sentiment, data=rock,family = binomial)
summary(fit)
```

```
##
## Call:
## glm(formula = genrerock ~ Rank + NumberWords + Sentiment, family = binomial,
##     data = rock)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.0083  -0.6984  -0.5881  -0.3856   2.2856
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.5589695  0.1693548  -3.301 0.000965 ***
## Rank         0.0026495  0.0018920   1.400 0.161416
## NumberWords -0.0031395  0.0004103  -7.651 1.99e-14 ***
## Sentiment   -1.8289528  0.5265700  -3.473 0.000514 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2198.4  on 2299  degrees of freedom
## Residual deviance: 2117.8  on 2296  degrees of freedom
## AIC: 2125.8
##
## Number of Fisher Scoring iterations: 5
```

```
probs = predict(fit,type="response")
prediction = predict(fit)
prediction = rep("Not Rock",nrow(rock))
prediction[probs > .5]="Rock"
real = recode(rock$genrerock,"1"="Rock","0"="Not Rock")
table(prediction,real)
```

```
##           real
## prediction Not Rock Rock
##   Not Rock     1876  424
```

```
unique(prediction)
```

```
## [1] "Not Rock"
```

This time, the model predicted "Rock" (1) one time, which was a false positive. I'll consider this model bust.

Predicting Audio Analysis

Now I am going to incorporate results from my Spotify API into these logisitc regressions. Hip-Hop is my favorite type of music, and tends to have pretty unique musical traits, so I'll use rap as a prediction variable out of non-pop songs.

```
aa = final_data[(final_data$Loudness != 9999) && (final_data$Tempo != 9999),]
```

```
fit = glm(genrepop ~ UniqueWords + Loudness:Tempo,data=aa,family = binomial)
summary(fit)
```

```
##
## Call:
## glm(formula = genrepop ~ UniqueWords + Loudness:Tempo, family = binomial,
##     data = aa)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3431  -0.9501  -0.8161   1.3586   2.0870
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)     9.611e-02  9.746e-02   0.986    0.324
## UniqueWords    -9.067e-03  9.554e-04  -9.490  < 2e-16 ***
## Loudness:Tempo  3.943e-09  7.872e-10   5.009 5.47e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4459.5  on 3480  degrees of freedom
## Residual deviance: 4353.9  on 3478  degrees of freedom
## AIC: 4359.9
##
## Number of Fisher Scoring iterations: 4
```

Both variables are signficantly different from 0 but their coefficients are so small that any change in the variable has negligible effect. It is interesting that unique words has a negetive coefficient, though. Maybe because Pop songs often repeat the same phrase or word often.

And the confusion table.

```
probs = predict(fit,type="response")
prediction = predict(fit)
prediction = rep("Not Pop",
                 nrow(aa[(aa$Loudness != 9999)&&(aa$Tempo != 9999),]))

prediction[probs > .5]="Pop"
real = recode(aa$genrepop,"1"="Pop","0"="Not Pop")
table(prediction,real)
```

```
##           real
## prediction Not Pop  Pop
##    Not Pop    2276 1172
##    Pop          24    9
```

```
# Percent Correct
mean(prediction==real)
```

```
## [1] 0.6564206
```

```
# False Positive Rate
20/23
```

```
## [1] 0.8695652
```

```
# True Positive Rate
2540/2560
```

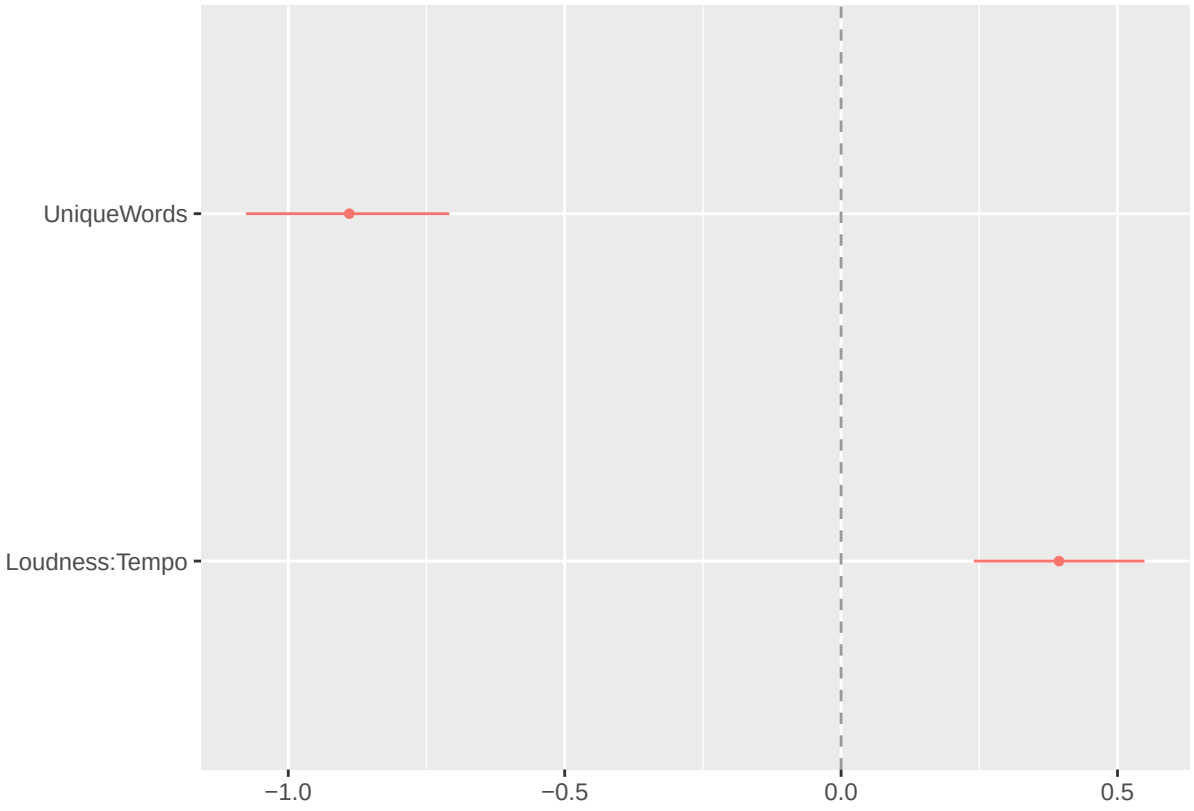
```
## [1] 0.9921875
```

For the sake of shortness, I ran a bunch of different combinations and this is the best I could predict. The model rarely predicts that a song is "Pop", but has a high false positive rate. Because the model mostly outputs "Not Pop" it has a high true positive rate - 66% of the data was "Not Pop".

Standard Errors

As I mentioned above, the coefficients are very small, though significantly different from zero.

```
dwplot(fit,vline = geom_vline(xintercept = 0, colour = "grey60", linetype = 2))
```



Probit Regression

Here is a probit regression for comparison.

```
fit = glm(genrepop ~ UniqueWords + Loudness:Tempo,
          data=aa[(aa$Loudness != 9999)&&(aa$Tempo != 9999),],
          family = binomial(link="probit"))


probs = predict(fit,type="response")
prediction = predict(fit)
prediction = rep("Not Pop",nrow(aa[(aa$Loudness != 9999)&&(aa$Tempo != 9999),]))
prediction[probs > .5]="Pop"
```

```
real = recode(aa$genrepop,"1"="Pop","0"="Not Pop")
table(prediction,real)
```

```
##           real
## prediction Not Pop  Pop
##    Not Pop    2279 1173
##    Pop          21    8
```

It seems that the coefficients in the probit regression are different from the logit, but still so small that they are mostly negligible.

Conclusions

I had hoped that the results with the new independent columns would be better predictors of genre, but it seems that my data is not well enough suited to predict anything from the data.

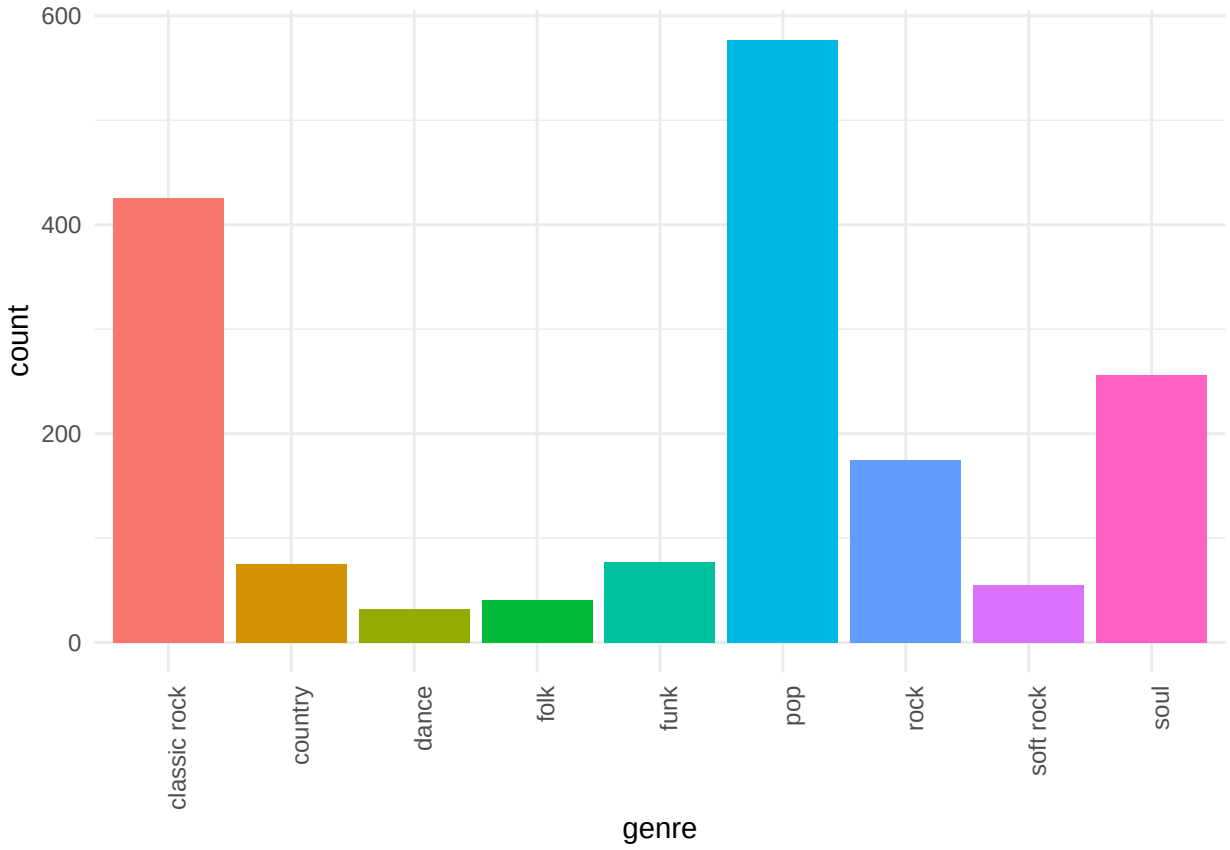
I'm not really sure where I'll go from here, but I think I can find ways to better predict genre or rank.

Report 3

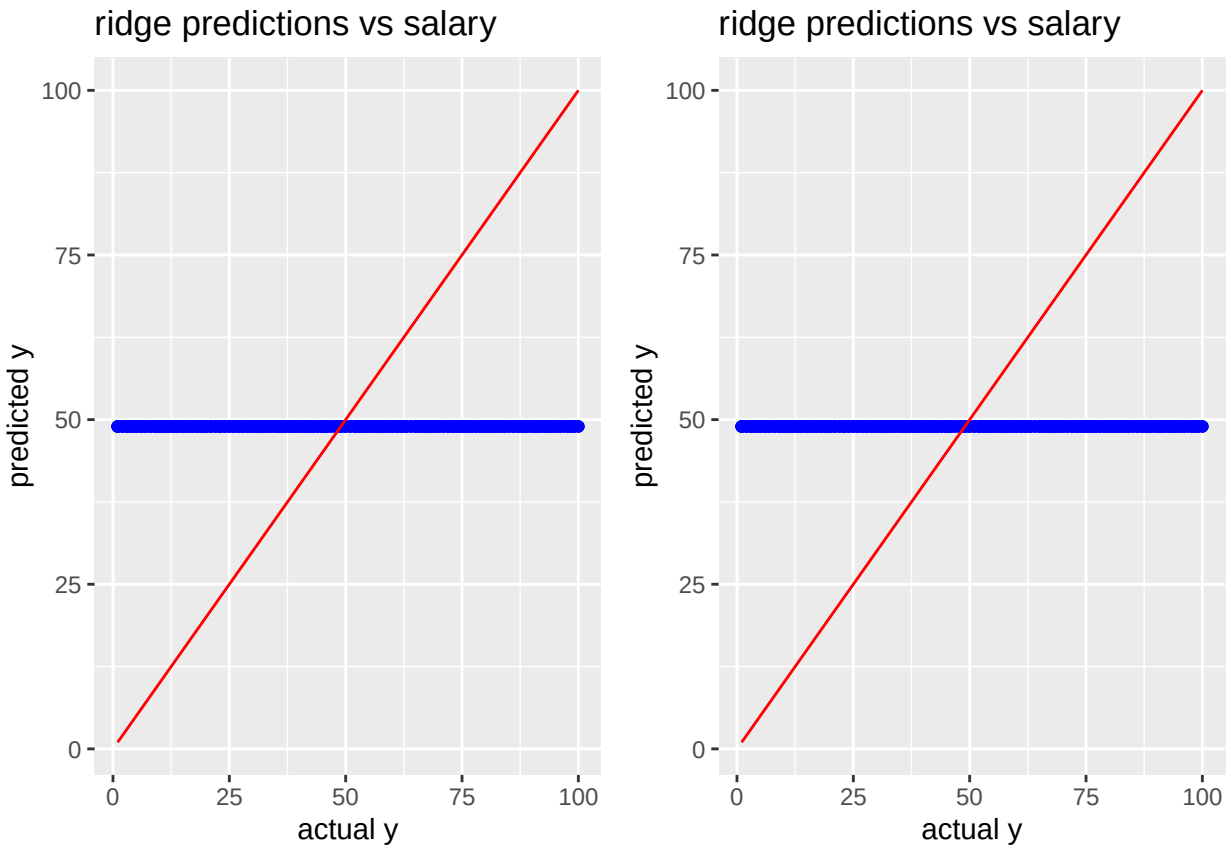
Introduction

This data has less than half of the observations as the complete dataset, but looking at the histogram of genres, we should have enough to perform classification of pop songs.

```
num_genre = current_data %>% group_by(genre) %>%
  summarise(count=n()) %>%
  select(genre,count)

genreplot = num_genre[num_genre$count > 20,] %>%
    ggplot(., aes(x=genre, y=count, fill=genre)) +
    geom_bar(stat="identity")+theme_minimal()+
    theme(axis.text.x = element_text(angle = 90, hjust = 1),
          legend.position = 'none')

genreplot
```



I already ran ridge and lasso models on a linear regression trying to predict Rank. Here's the prediction results plotted against the actual for both.



Obviously, Rank is not going to work for any kind of regression. I think it's due the to fact that Rank is uniformly distributed. The models can minimize their RSS and objective functions by setting the intercept to 50 and the coefficients to such small numbers that the prediction will always be around 50. It's unfortunate, hopefully non-linear models will work better.

Either way, I used a classification variable for the ridge and lasso models.

Question 1. Validation and Training Sets

I let my training data include the majority of the data. Here's the distribution of genres among the two sets.

```
train_ = current_data %>%
  select(genre,Rank,c(8:18)) %>%
  sample_frac(0.6)
test_ = current_data %>%
  select(genre,Rank,c(8:18)) %>%
  setdiff(train_)
par(mfrow=c(2,1))

train_genres = train_ %>% group_by(genre) %>%
  summarise(count=n()) %>%
  filter(count > 20) %>%
    ggplot(., aes(x=genre, y=count, fill=genre)) +
    geom_bar(stat="identity")+theme_minimal()+
    theme(axis.text.x = element_text(angle = 90, hjust = 1),
          legend.position = 'none') +
    ggtitle("Distribution of Genres in Train Data")
```

```
test_genres = test_ %>% group_by(genre) %>%
  summarise(count=n()) %>%
  filter(count > 20) %>%
    ggplot(., aes(x=genre, y=count, fill=genre)) +
    geom_bar(stat="identity")+theme_minimal()+
    theme(axis.text.x = element_text(angle = 90, hjust = 1),
          legend.position = 'none') +
    ggtitle("Distribution of Genres in Test Data")

grid.arrange(train_genres, test_genres, ncol=2)
```

This following chunk creates a binary column for whether a song is of genre "Pop" or not. I use "-1" instead of "0" for better results.

```
train_$IsPop = ifelse(train_$genre == "pop",1,-1)
test_$IsPop = ifelse(test_$genre == "pop",1,-1)
```

Question 2. Ridge and Lasso Regressions.

Given that Rank proves to be unpredictable through linear models, I will preform ridge and lasso regressions on the binary variable, IsPop.

Ridge Regression

Question 2a. Tuning the model.

```
x = model.matrix(IsPop~., train)[,-1]
y = train$IsPop
grid = 10^seq(10, -2, length = 100)

ridge.cv = cv.glmnet(x,y,alpha=0)
```

After running the cross-validated model, the λ that minimizes the RSS/objective function is:

```
## [1] 0.1341999
```

Question 2b. The Truly Best Lambda

In practice, we use the best lambda 1 standard error away from the minimum λ.

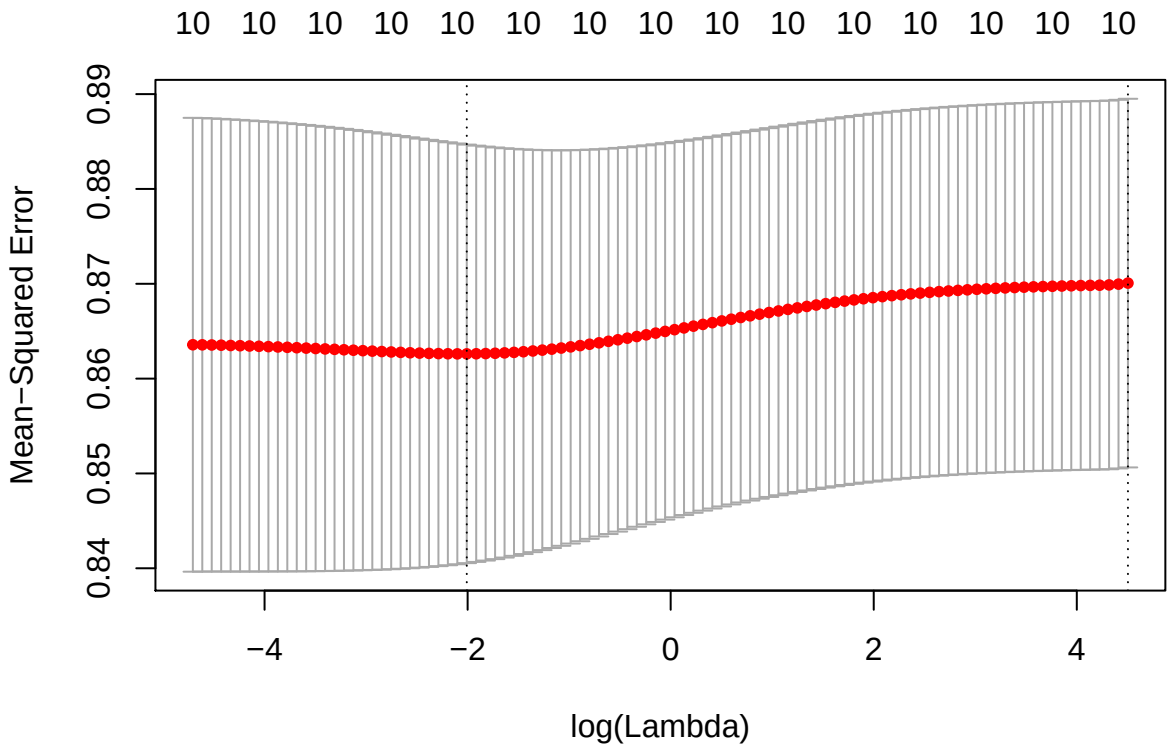
```
bestlam = ridge.cv$lambda.1se
bestlam
```

```
## [1] 90.37225
```

Honestly, it strikes me as odd that the two λ's are so far apart. I wonder why.

Question 2c. Cross-Validation and the Truly Best Lambda

```
plot(ridge.cv)
```

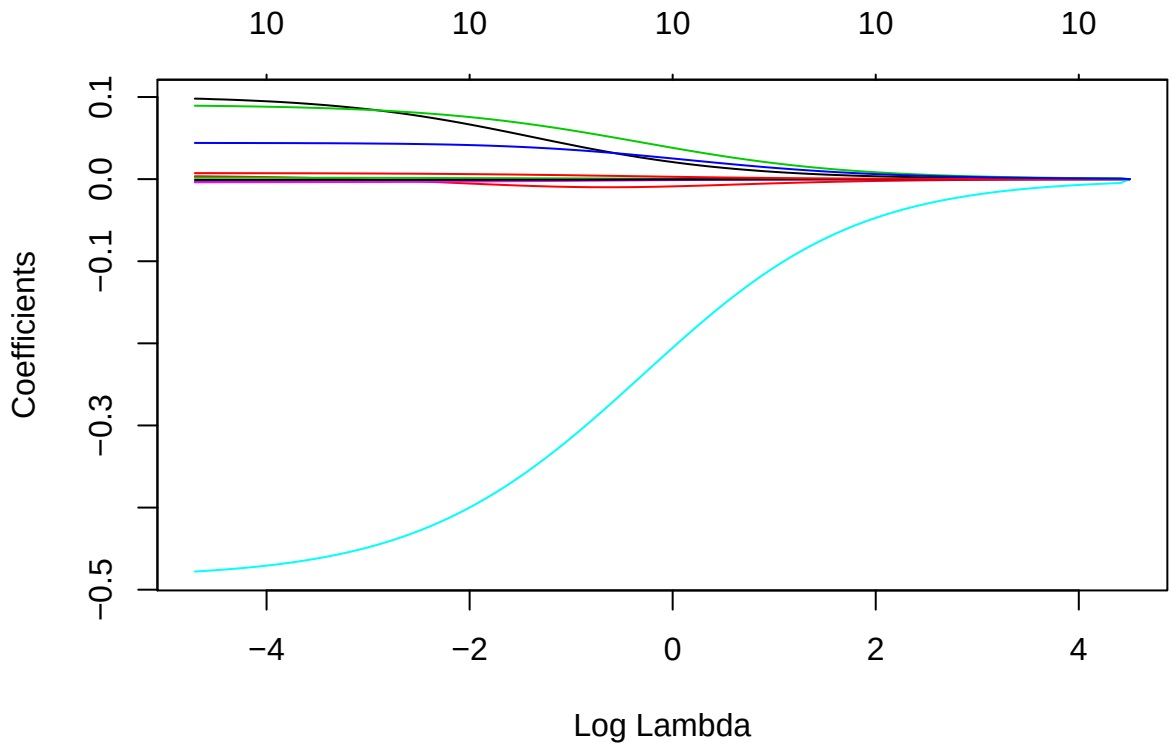


Well that would be why.

Question 2d. Coefficients

Running a normal Ridge Regression, here's how the coefficients vary as λ increases:

```
ridge = glmnet(x,y,alpha=0)
plot(ridge,xvar="lambda")
```



Question 2e. Coefficients of the Best Lambda

And here are the coefficients for the λ one SE from the minimum λ:

```
predict(ridge.cv, type = "coefficients", s = bestlam)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                                    1
## (Intercept)          -3.625349e-01
## AverageWordLength     2.882727e-38
## AverageRhymeLength   -2.024538e-38
## NumberWords           9.566828e-40
## UniqueWords           3.575618e-40
## Sentiment            -4.263566e-37
## Loudness             -2.057485e-39
## Tempo                -1.188563e-39
## Key                   4.788017e-39
## Mode                  7.456503e-38
## time_signature        5.621368e-38
## feature               .
```

Question 2f. Testing the Model

Using the Test subset to generate predictions from the model, we can generate the truth table of the model.

```
x_train = model.matrix(IsPop~., train)[,-1]
x_test = model.matrix(IsPop~., test)[,-1]
y_train = train$IsPop
y_test = test$IsPop


ridge_pred = predict(ridge.cv, s = bestlam, newx = x_test)
prediction = rep("Not Pop",nrow(ridge_pred))
prediction[ridge_pred > 0]="Pop"
real = recode(test$IsPop,"1"="Pop","-1"="Not Pop")
table(prediction,real)
```

```
##           real
## prediction Not Pop Pop
##    Not Pop     481 235
```

If you remember the second report, I ran into this issue, where the model only predicts "Not Pop". Hopefully Lasso might prove more helpful.

In comparison to my Logit regressions, I get the same outcome, so comparing them seems uneccessary.

Lasso Regression

Using the same binary column, IsPop, let's see if a Lasso model proves more useful.

Question 2a. Tuning the model.

```
x = model.matrix(IsPop~., train)[,-1]
y = train$IsPop
grid = 10^seq(10, -2, length = 100)

lasso.cv = cv.glmnet(x,y,alpha=1)
```

After running the cross-validated model, the λ that minimizes the RSS/objective function is:

```
## [1] 0.001992826
```

Question 2b. The Truly Best Lambda

In practice, we use the best lambda 1 standard error away from the minimum λ.

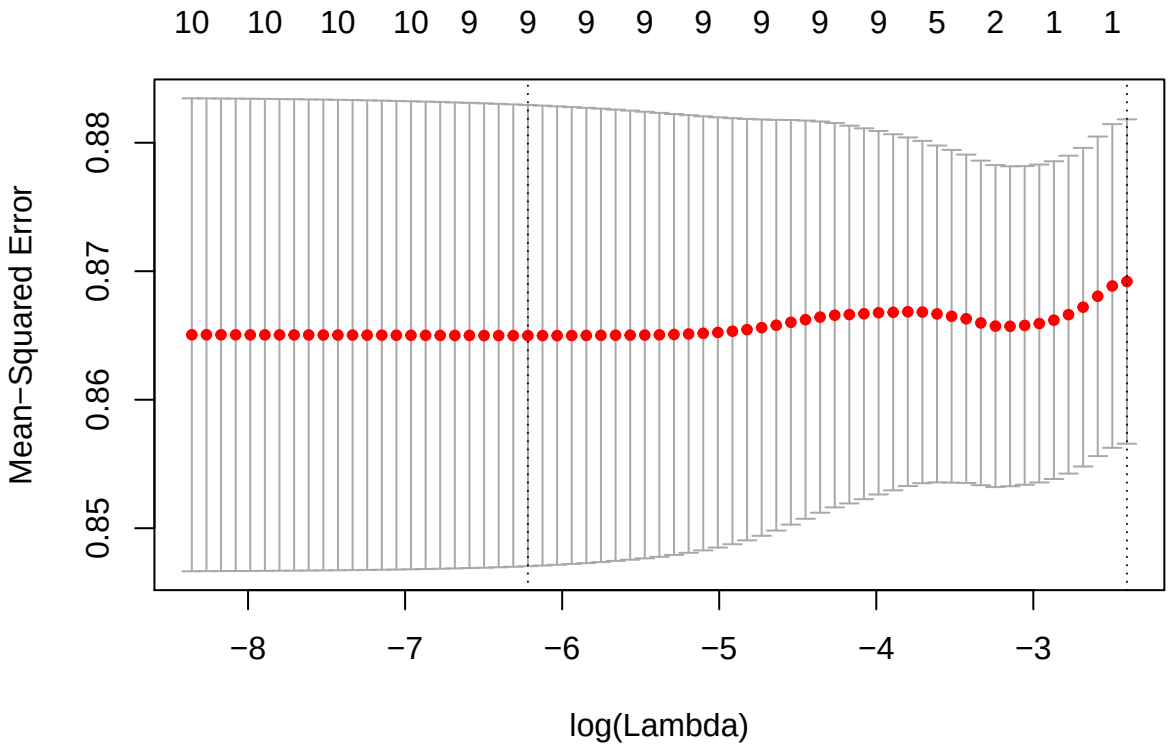
```
bestlam = lasso.cv$lambda.1se
bestlam
```

```
## [1] 0.09037225
```

Again, these λs are strange, this time they are incredibly close together, but very small.

Question 2c. Cross-Validation and the Truly Best Lambda

```
plot(lasso.cv)
```

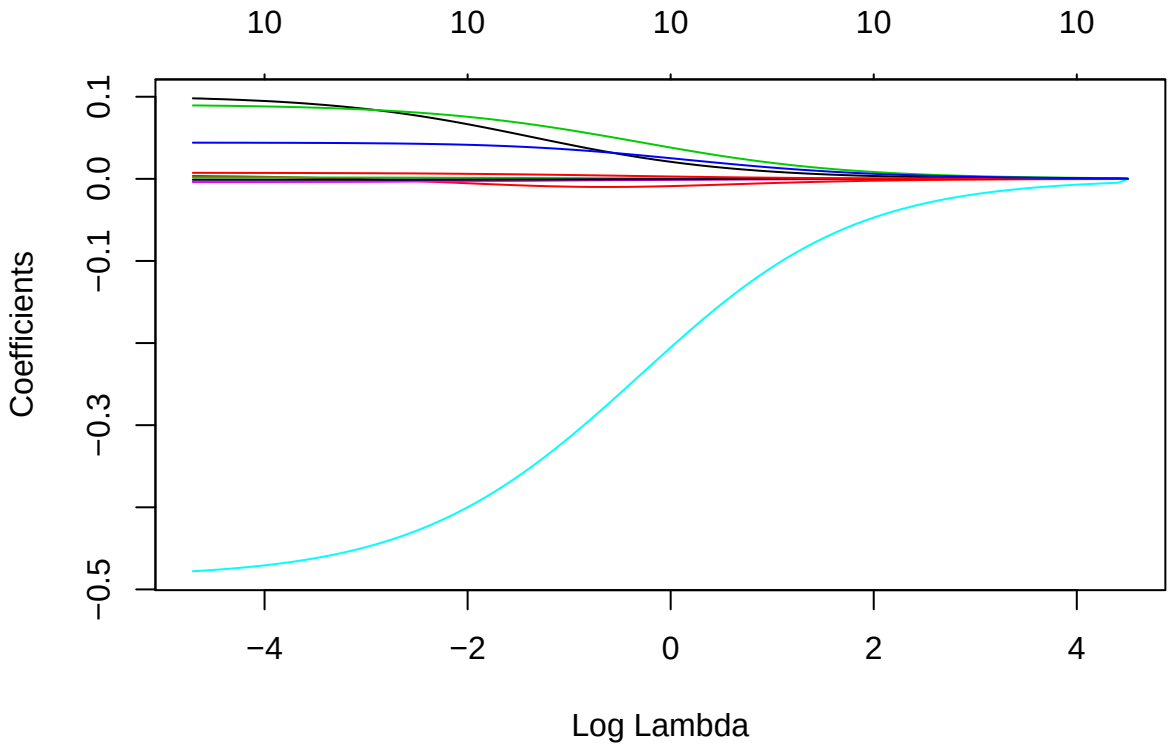


I'm curious why there is a bump just before the minimum λ, but this plot does explain why the λs are so small, the minimizing λ is $\sim e^{-3}$ and the λ 1 SE from the minimum is not much larger.

Question 2d. Coefficients

Running a normal Ridge Regression, here's how the coefficients vary as λ increases:

```
lasso = glmnet(x,y,alpha=1)
plot(ridge,xvar="lambda")
```



Question 2e. Coefficients of the Best Lambda

And here are the coefficients for the λ one SE from the minimum λ:

```
predict(lasso.cv, type = "coefficients", s = bestlam)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                              1
## (Intercept)        -0.3625349
## AverageWordLength   .
## AverageRhymeLength  .
## NumberWords         .
## UniqueWords         .
## Sentiment           .
## Loudness            .
## Tempo               .
## Key                 .
## Mode                .
## time_signature      .
## feature             .
```

Well. That's... I'm not sure what I expected, but this isn't to far from it. I'll finish the rest of the section, but everything else I do from here will be meaningless (because the model only includes the intercept).

Question 2f. Testing the Model

Using the Test subset to generate predictions from the model, we can generate the truth table for the model.

```
x_train = model.matrix(IsPop~., train)[,-1]
x_test = model.matrix(IsPop~., test)[,-1]
y_train = train$IsPop
y_test = test$IsPop


lasso_pred = predict(lasso.cv, s = bestlam, newx = x_test)
lasso_prediction = rep("Not Pop",nrow(lasso_pred))
lasso_prediction[lasso_pred > 0]="Pop"
real = recode(test$IsPop,"1"="Pop","-1"="Not Pop")
table(lasso_prediction,real)
```

```
##                 real
## lasso_prediction Not Pop Pop
##          Not Pop     481 235
```

This is actually funny. The coefficients of the ridge model are so negligible that if you didn't factor them in, the predictions wouldn't change. I know this because we can run the following code to show the predictions are the same for the intercept-only lasso model and the ridge model.

```
unique(prediction == lasso_prediction)
```

```
## [1] TRUE
```

So linear models prove useless for predicting Rank and IsPop. Next, our first non-linear model, binary trees!

Question 3. Decision Tree

I've run both of the following trees many times. Sometimes the algorithm never splits and leaves me with an empty tree, other times it splits by 5 or 6 variables. I'll write the following assuming the trees are non-empty.

The Tree

First to create our tree. I'm going to try to predict IsPop.

Question 3a. Pruning the Tree

```
tree_train = train_ %>% select(-c(Rank,genre))
tree_train$IsPop = as.factor(ifelse(tree_train$IsPop >= 0, "Pop","Not Pop"))
tree_test = test_ %>% select(-c(Rank,genre))
tree_test$IsPop = as.factor(ifelse(tree_test$IsPop >= 0, "Pop","Not Pop"))

t2 = tree(IsPop ~ .,data = tree_train)
summary(t2)
```

```
##
## Classification tree:
## tree(formula = IsPop ~ ., data = tree_train)
## Variables actually used in tree construction:
## [1] "NumberWords" "Tempo"       "UniqueWords"
## Number of terminal nodes:  4
## Residual mean deviance:  1.213 = 1296 / 1069
## Misclassification error rate: 0.302 = 324 / 1073
```

Hey this model has a split! Let's see if it can predict anything!

```
tree_pred = predict(t2, tree_test, type = "class")
table(tree_pred, tree_test$IsPop)
```

```
##
## tree_pred Not Pop Pop
##   Not Pop     433 204
##   Pop          48  31
```

Not terrible, I think.

Question 3b. Plotting the Pruned Tree

```
t2_pruned = tryCatch(cv.tree(IsPop ~ .,data=tree_train),
         warning=NULL,
         error = function(e) {print(e)},
         finally= NULL)
```

```
## <simpleError in cv.tree(IsPop ~ ., data = tree_train): not legitimate tree>
```

```
if(length(t2_pruned) <= 2 | is.null(t2_pruned)) {
  print("Tree is illigitimate, whatever that means")
} else {
  plot(t2_pruned) + text(t2_pruned,pretty = 0)
}
```

```
## [1] "Tree is illigitimate, whatever that means"
```

I'm not sure, but I keep getting an error saying that the tree is not a "legitimate" tree. I'll use the if statement to catch it, hopefully the tree is legitimate.

I've been getting different splits on the second tree everytime I run, so I'll still do the extra credit, even if it's seemingly pointless. Hopefully reading this has been more fun than writing it haha. :(.

Extra Credit

```
tree_gen = function(data,indices) {
  d = data[indices,]
  t = tree(IsPop ~ .,data=d)
  t_pred = predict(t,d,type="class")
  correct = sum(t_pred == d[,"IsPop"])
  return(correct/nrow(d))
}
```

Using the above function as the statistic in the bootstrap, we can generate 100 trees to find the aggregated error rate.

```
d = current_data %>%
  mutate(IsPop = ifelse(genre == "pop","Pop","Not Pop")) %>%
  select(IsPop,c(8:18))
d$IsPop = as.factor(d$IsPop)
boot = boot::boot(d,statistic=tree_gen,R=100)
boot$t0
```

```
## [1] 0.6774734
```

So a bootstrap of my model has an error rate of 67%. I'm actually shocked. Compare this error rate to my single tree from above, and it is signficantly better. I'm not convinced, however, that the 67% of correct predictions are due to the model being correct – I think that the sampling with replacement leads to higher proportions of "Not Pop" songs and the model still isn't predicting anything but "Not Pop". I'm not sure how to pursue this, I'll probably leave it for next time.

Conclusions

I'm not sure how much hope I have for random forest or bagging, but hopefully they prove more helpful than everything above. I was hoping that this project would have more interesting results, but it seems that this kind of machine learning is not proficient at these kind of estimations.