BrazilSpeaks

Gabriel Saruhashi

3/17/2019

Intro

On April 1, 1964, the military organized a coup d'état that overthrew the government of president João Goulart. That day marked the beginning of the Military Dictatorship that lasted for twenty-one years. Under the pretext of eliminating the growing Communist threaT suppressed freedom of speech and imposed rigorous censorship over the all forms of media. In the late 60s, with the popularization of television and radio stations, music began to have a lot of influence over society and, for this reason, it was heavily monitored by the regime's censors. On the one hand, there was a group of musicians that simply conformed to the oppressive rules of the regime. Inspired by the soft rock melodies by the Beatles, they avoided political themes and made fortunes composing songs about love and trivial, middle-class concerns. Yet, on the other hand, a group of musicians stood out in the fight against oppression. Through their music, they conveyed a message of criticism against the regime. Their "protest music" denounced blatant social injustices, mobilized political passions, praised the individual and collective heroes who fought the oppressors.

DATA

Data Scraping & Collection

To collect the data, It requires

I compiled two Spotify playlists, one for each class of music. Through the Spotify API, I obtained key features of each song, such as speechness, danceability and energy, that are measured in a scale of 0.0 to 1.0 (Figure 3). However, Spotify does not directly provide the lyrics for each of the songs. To circumvent this limitation, I built a parallel pipeline that, given a song name ands its author, scrapes song lyrics from Genius and Vagalume, two well-known music platform that provide lyrics and song annotations. The procedure yielded a corpus of 280 songs equally divided in the two categories: 140 censored and 140 uncensored songs.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from spotipy.oauth2 import SpotifyClientCredentials
import spotipy
import json
import requests
from bs4 import BeautifulSoup
import pandas as pd
import pprint
import time
from nltk.stem import RSLPStemmer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import os
import re

def setEnvironmentVariables():
    os.environ['SPOTIPY_CLIENT_ID'] = 'c894a126681b4d97a8ccb0cd4a1e0de1'
```

```python
    os.environ['SPOTIPY_CLIENT_SECRET'] = 'ebf185aaf47e40ab841246986fc7483d'
    os.environ['SPOTIPY_REDIRECT_URI'] = 'https://localhost:8080'
    print('Successfully set the environment variables')

def requestSongInfo(song_title, artist_name):
    base_url = 'https://api.genius.com'
    headers = {'Authorization': 'Bearer ' + 'ORIKjAuJB6gohq-1r-w7FzG7W3FcgsL2ZwSRWjUdLLHOE31lUt6T8otW-J
    search_url = base_url + '/search'
    data = {'q': song_title + ' ' + artist_name}
    response = requests.get(search_url, data=data, headers=headers)

    return response

def scrapeSongURL(url):
    print("scraping {}".format(url))
    page = requests.get(url)
    html = BeautifulSoup(page.text, 'html.parser')
    lyrics = html.find('div', class_='lyrics').get_text()

    return lyrics

# Preprocessing of the Lyrics
def preprocessLyrics(sentence):
    # stemmer=RSLPStemmer()

    sentence = sentence.lower()
    # remove all the annotations (e.g '[refrão 1] Bla bla')
    sentence = re.sub(r'[\(\[].*?[\)\]]', "", str(sentence))

    # get Portuguese stopwords
    file_stop = open("pt_stopwords.txt")
    body_stop = file_stop.read()
    stop = body_stop.split()

    token_words = word_tokenize(sentence)
    processed_sentence=[]

    for word in token_words:
        if word not in stop:
            processed_sentence.append(word)
            # stem_sentence.append(stemmer.stem(word))
            processed_sentence.append(" ")

    # remove all the annotations within [] and ()

    return "".join(processed_sentence)

def extractLyrics(song_title, artist_name):
    # Search for matches in request response
    response = requestSongInfo(song_title, artist_name)
    json = response.json()
    remote_song_info = None
```

```python
    for hit in json['response']['hits']:
        if artist_name.lower() in hit['result']['primary_artist']['name'].lower():
            remote_song_info = hit
            break

    # Extract lyrics from URL if song was found
    if remote_song_info:
        song_url = remote_song_info['result']['url']
        lyrics = scrapeSongURL(song_url)
        lyrics = lyrics.replace('\n', ' ')
        lyrics = preprocessLyrics(lyrics)

        return lyrics
    else:
        print("Could not find lyrics for given artist and song title")
        return ""

def getSpotifySongFeatures(uri):
    song_features = sp.audio_features(uri)
    song_features = song_features[0]

    extra_fields = ["track_href", "uri", "analysis_url", "type"]

    for field in extra_fields:
        song_features.pop(field)

    return song_features

def getSpotifyArtistInfo(artist_id):
    artist = {}

    info = sp.artist(artist_id)

    artist["artist_genres"] = info["genres"][0]
    artist["artist_name"] = info["name"]
    if info["images"]:
        artist["artist_photo"] = info["images"][0]["url"]
    else:
        artist["artist_photo"] = ""
    artist["artist_popularity"] = info["popularity"]
    artist["artist_sp_followers"] = info["followers"]["total"]

    return artist

def processSpotifyPlaylistCSV(uri, csv_filepath, song_class):
    start_time = time.time()

    username = uri.split(':')[2]
    playlist_id = uri.split(':')[4]

    # get the relevant playlist
    results = sp.user_playlist(username, playlist_id)
```

```
    tracks = results["tracks"]["items"]

    # define main data frame that will store
    df = pd.DataFrame()
    index = 0
    for obj in tracks:
        track = obj["track"]
        song = {}

        # preprocessed song name
        song_name = re.split(r' -| \(', track["name"])[0]

        # song["artist"] = artist
        song["song_sp_uri"] = track["uri"]
        song["song_name"] = song_name
        song["song_isrc"] = track["external_ids"]["isrc"]
        song["song_popularity"] = track["popularity"]
        song_features = getSpotifySongFeatures(track["uri"])

        artist_info = getSpotifyArtistInfo(track["artists"][0]["id"])
        song["song_lyrics"] = extractLyrics(song["song_name"], artist_info["artist_name"])
        song["class"] = song_class

        # concatenating all dictionaries
        song = {**song, **song_features, **artist_info}

        df = pd.concat([df, pd.DataFrame(song, index=[index])])
        index += 1

    print("Scraping process took {} s. Now storing intermediate results for this class of music".format
    df.to_csv(csv_filepath)

    return df

# Uncomment this section if you'd like to start the datascraping script
'''
PROTEST_URI = 'spotify:user:gabriel_saruhashi:playlist:4Tp4QcTk9rNikjmaDg5VxJ'
JOVEM_GUARDA_URI = 'spotify:user:gabriel_saruhashi:playlist:1JZoMCGiAKcXrgBzbKW931'
PROTEST_CLASSNAME = "Protest"
JOVEM_GUARDA_CLASSNAME = "Jovem Guarda"
setEnvironmentVariables()

client_credentials_manager = SpotifyClientCredentials()
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

# create csv with data from spotify
protest_df = processSpotifyPlaylistCSV(PROTEST_URI, "protest.csv", "Protest")
jovem_guarda_df = processSpotifyPlaylistCSV(JOVEM_GUARDA_URI, "jovem_guarda.csv", "Jovem Guarda")

# store final output
res_df = pd.concat([protest_df, jovem_guarda_df])
res_df.to_csv("brz_dictatorship.csv")
'''
```

Overview of the data

Upon loading the data, we observe the following structure: * song_sp_uri (chr): a unique identifies song in the spotify platform * song_name (chr): the name of the song * song_isrc (chr): the International Standard Recording Code for the song * song_popularity (int) : provided by the Spotify API, "The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are" * song_lyrics (chr): the lyrics of the song scraped from Genius and Vagalume * class (chr): the class of the song (either protest or Young Guard) according to the definition presented in the intro * danceability (num): provided by the Spotify API, "a value of 0.0 is least danceable and 1.0 is most danceable." * energy (num): provided by the Spotify API, "energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity" * key (int): provided by the Spotify API, "the estimated overall key of the track" * loudness (int): provided by the Spotify API, "the overall loudness of a track in decibels (dB)" * mode * speechiness (num): provided by the Spotify API, "float Speechiness detects the presence of spoken words in a track" * acousticness (num): provided by the Spotify API, "A confidence measure from 0.0 to 1.0 of whether the track is acoustic" * instrumentalness (num): provided by the Spotify API, "predicts whether a track contains no vocals" * liveness (num): provided by the Spotify API, "detects the presence of an audience in the recording. . Higher liveness values represent an increased probability that the track was performed live" * valence (num): provided by the Spotify API, "a measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track" * tempo (num): provided by the Spotify API, "the overall estimated tempo of a track in beats per minute" * id (chr): the Spotify ID for the artist * duration_ms: provided by the Spotify API, "the duration of the track in milliseconds" * time_signature (int) * artist_genres (chr): provided by the Spotify API, "a list of the genres the artist is associated with * artist_name (chr): the name of the artist * artist_photo: (chr): url to the photo of the artist * artist_popularity (int): provided by the Spotify API,"the value will be between 0 and 100, with 100 being the most popular. " * artist_sp_followers (int): 542214 542214 542214 299597 829961 532021 542214 16490 2440436 299597

```
## [1] "Number of dimensions in our dataset (row, col):"
```

```
## [1] 200  26
```

Create corpus for text mining

```r
library(tm)
```

```
## Loading required package: NLP
```

```r
print("Creating corpus by collapsing together both protest music and Young Guard music")
```

```
## [1] "Creating corpus by collapsing together both protest music and Young Guard music"
```

```r
jg <- paste(music$song_lyrics[music$class=="Jovem Guarda"], collapse = '')
protest <- paste(music$song_lyrics[music$class=="Protest"], collapse = '')
docs <- Corpus(VectorSource(c(jg, protest)))
str(docs)
```

```
## List of 2
##  $ 1:List of 2
##   ..$ content: chr "terrivel bom parar desse jeito provocar voce nao sabe onde venho terrivel vou di
##   ..$ meta   :List of 7
##   .. ..$ author       : chr(0)
##   .. ..$ datetimestamp: POSIXlt[1:1], format: "2019-05-07 01:17:40"
##   .. ..$ description  : chr(0)
##   .. ..$ heading      : chr(0)
##   .. ..$ id           : chr "1"
##   .. ..$ language     : chr "en"
##   .. ..$ origin       : chr(0)
##   .. ..- attr(*, "class")= chr "TextDocumentMeta"
```

```
##   ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
##  $ 2:List of 2
##   ..$ content: chr "pai , afasta mim calice pai , afasta mim calice pai , afasta mim calice vinho ti
##   ..$ meta   :List of 7
##   .. ..$ author       : chr(0)
##   .. ..$ datetimestamp: POSIXlt[1:1], format: "2019-05-07 01:17:40"
##   .. ..$ description  : chr(0)
##   .. ..$ heading      : chr(0)
##   .. ..$ id           : chr "2"
##   .. ..$ language     : chr "en"
##   .. ..$ origin       : chr(0)
##   .. ..- attr(*, "class")= chr "TextDocumentMeta"
##   ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
##  - attr(*, "class")= chr [1:2] "SimpleCorpus" "Corpus"
```

Data Cleaning

Although the song features supplied by the Spotify API were already normalized, I had to perform some preprocessing of the lyric. First, I removed stopwords (e.g 'me', 'I', etc.) from the dataset given that they are so common in the language that their informational value is near zero. Second I cleaned up the Genius lyrics by removing the annotations, punctuations and number.

```
# Load
library("tm")
library("SnowballC")
library("wordcloud")
```

```
## Loading required package: RColorBrewer
```

```
library("RColorBrewer")

# Remove numbers
docs <- tm_map(docs, removeNumbers)
# Remove Portuguese common stopwords
docs <- tm_map(docs, removeWords, stopwords("portuguese"))
# Remove punctuations
docs <- tm_map(docs, removePunctuation)
# Eliminate extra white spaces
docs <- tm_map(docs, stripWhitespace)
# Remove your own stop word
# specify your stopwords as a character vector
docs <- tm_map(docs, removeWords, c("mim", "pra", "vai"))
```

Descriptive Plots & Summary Information

Lyric Analysis

Inspired by the analysis conducted by (http://www.sthda.com/english/wiki/text-mining-and-word-cloud-fundamentals-in-r-5-si
First, I plotted a word cloud with the most frequent words for each class of music. Uncensored music had much more positive lyrics, with words such as love, romance and joy standing out, whereas protest music had more descriptive words such as violence, blood, etc.

```r
# Document matrix is a table containing the frequency of the words. Column names are words and row name
dtm_jg <- TermDocumentMatrix(docs[1])
m <- as.matrix(dtm_jg)
v <- sort(rowSums(m),decreasing=TRUE)
d_jg <- data.frame(word = names(v),freq=v)
head(d_jg, 10)
```

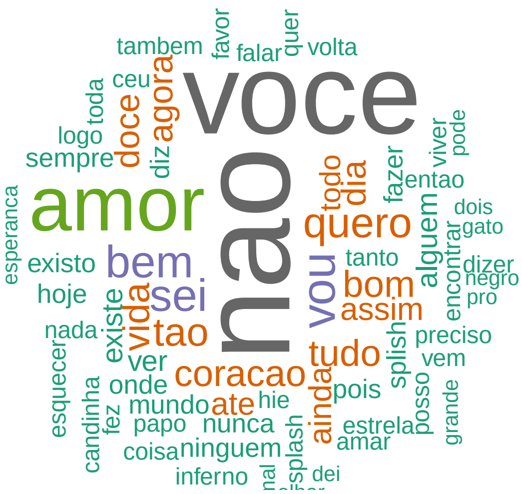
```
##            word freq
## nao         nao  268
## voce       voce  238
## amor       amor  146
## vou         vou   78
## bem         bem   73
## sei         sei   68
## quero     quero   63
## tao         tao   55
## coracao coracao   54
## tudo       tudo   54
```

```r
dtm_protest <- TermDocumentMatrix(docs[2])
m <- as.matrix(dtm_protest)
v <- sort(rowSums(m),decreasing=TRUE)
d_protest <- data.frame(word = names(v),freq=v)
head(d_protest, 10)
```

```
##        word freq
## nao     nao  416
## voce   voce  179
## tudo   tudo   94
## gente gente   74
## dia     dia   71
## sera   sera   64
## todos todos   64
## amor   amor   60
## faz     faz   59
## quero quero   58
```

```r
set.seed(1234)
wordcloud(words = d_protest$word, freq = d_protest$freq, min.freq = 15,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))
```

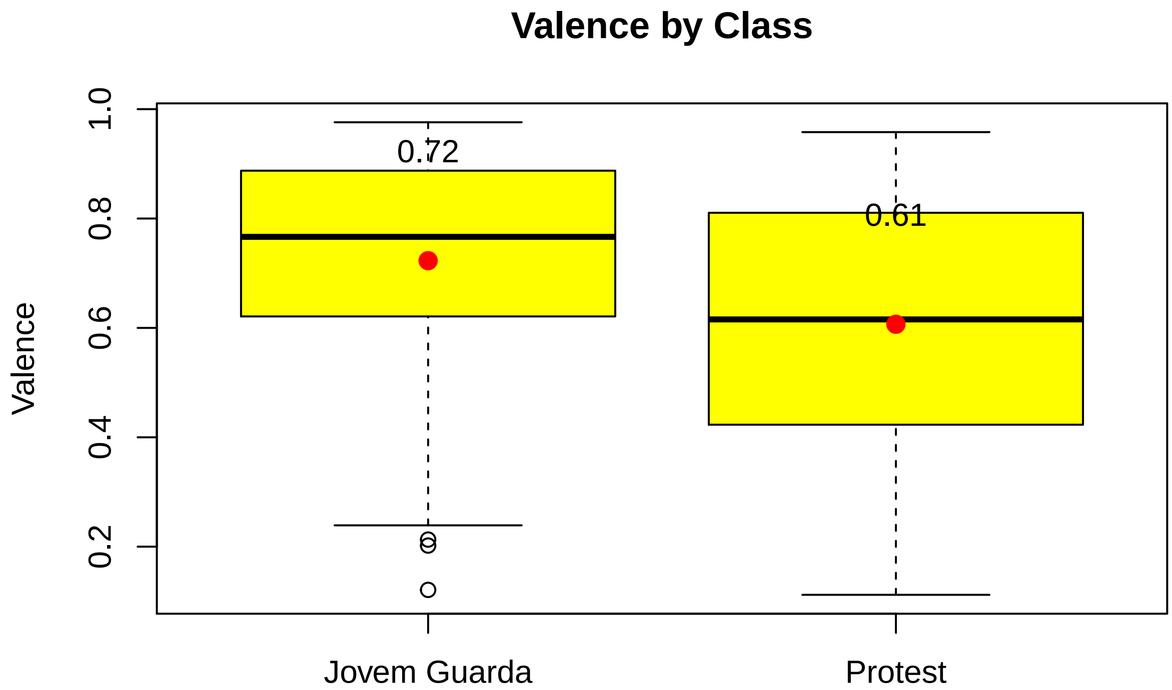

```r
wordcloud(words = d_jg$word, freq = d_jg$freq, min.freq = 15,
          max.words=200, random.order=FALSE, rot.per=0.35,
          colors=brewer.pal(8, "Dark2"))
```

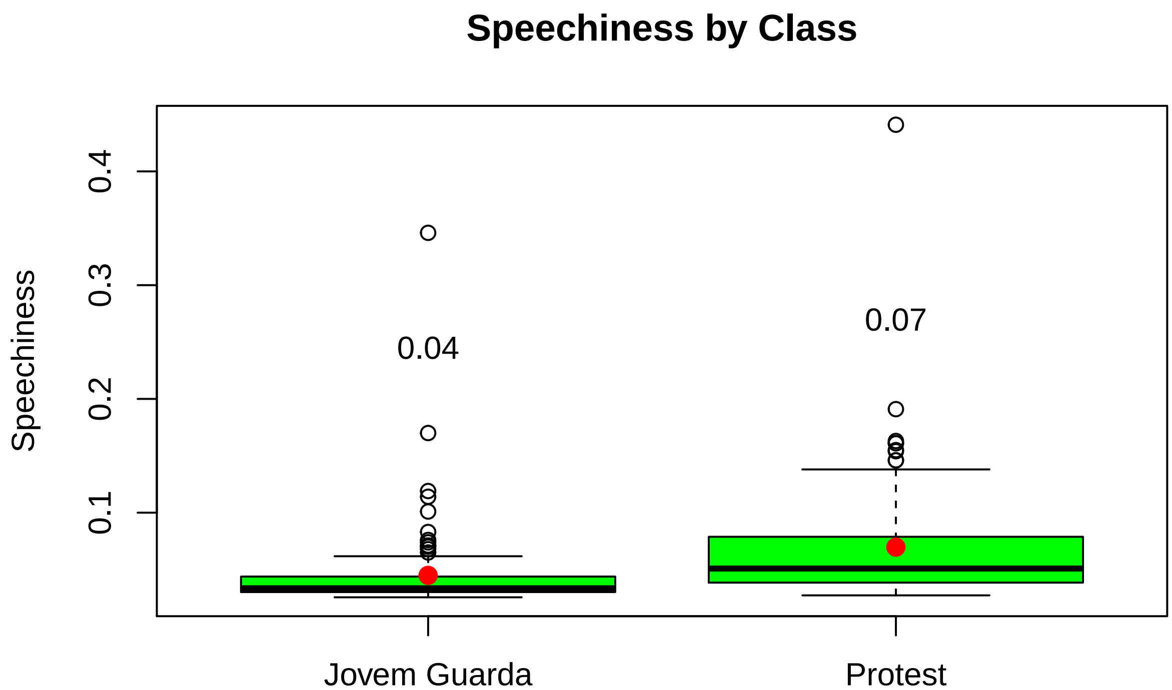


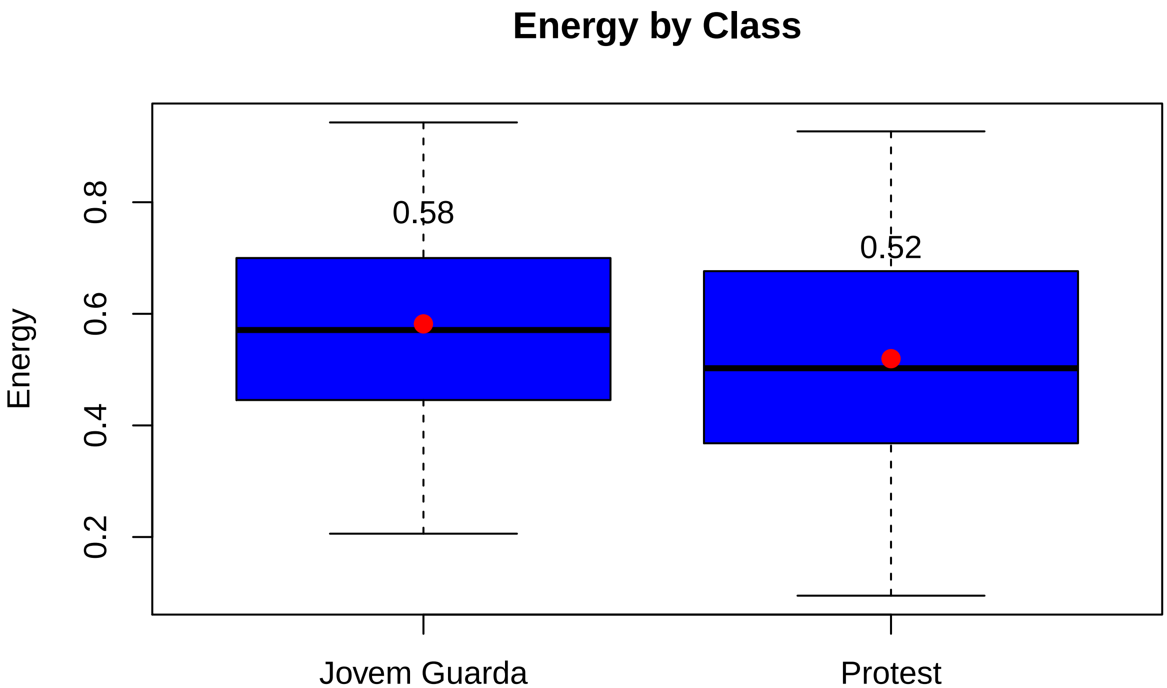
Then I performed ANOVA across four main song features, namely speechiness, energy, danceability and valence. As I imagined, protest music had higher speechiness given that the protest musicians prioritized the content of the message over form or harmonic features, whereas uncensored music had higher valence, danceability and energy. These characteristics were also in line with the insights gained from the historical study given that the Young Guard were known for their sappy songs that were popular in parties and bars (Table 1). All p-values were significant ($p < 0.05$).

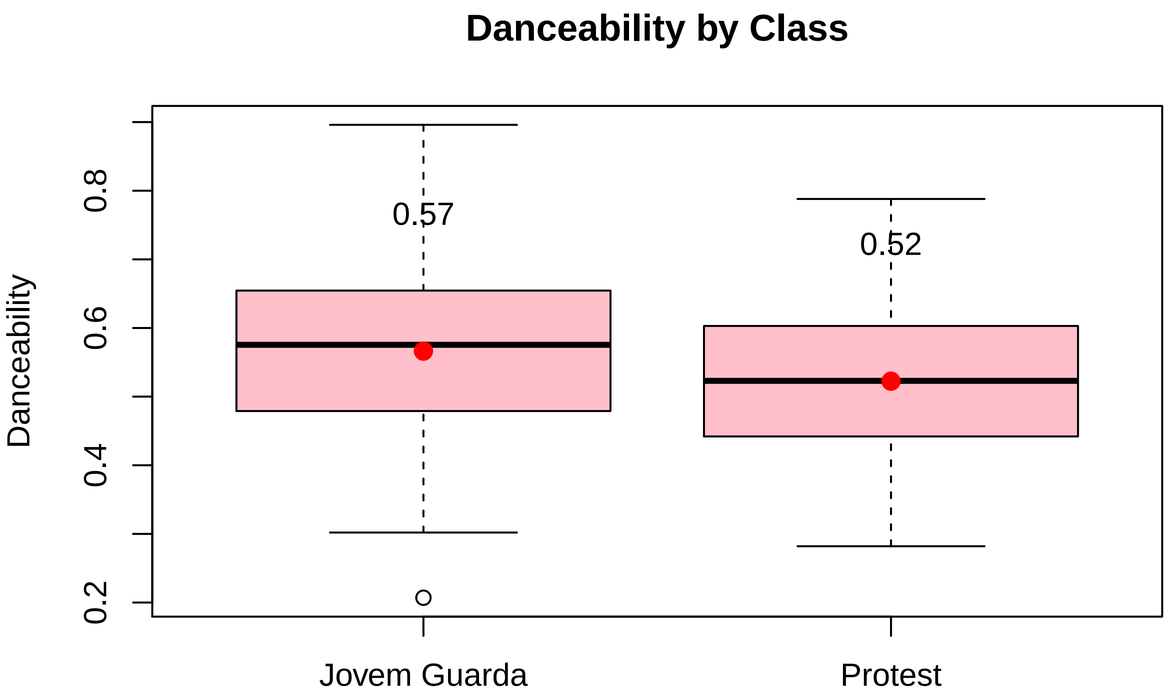
```
##
## Attaching package: 'ggplot2'
```

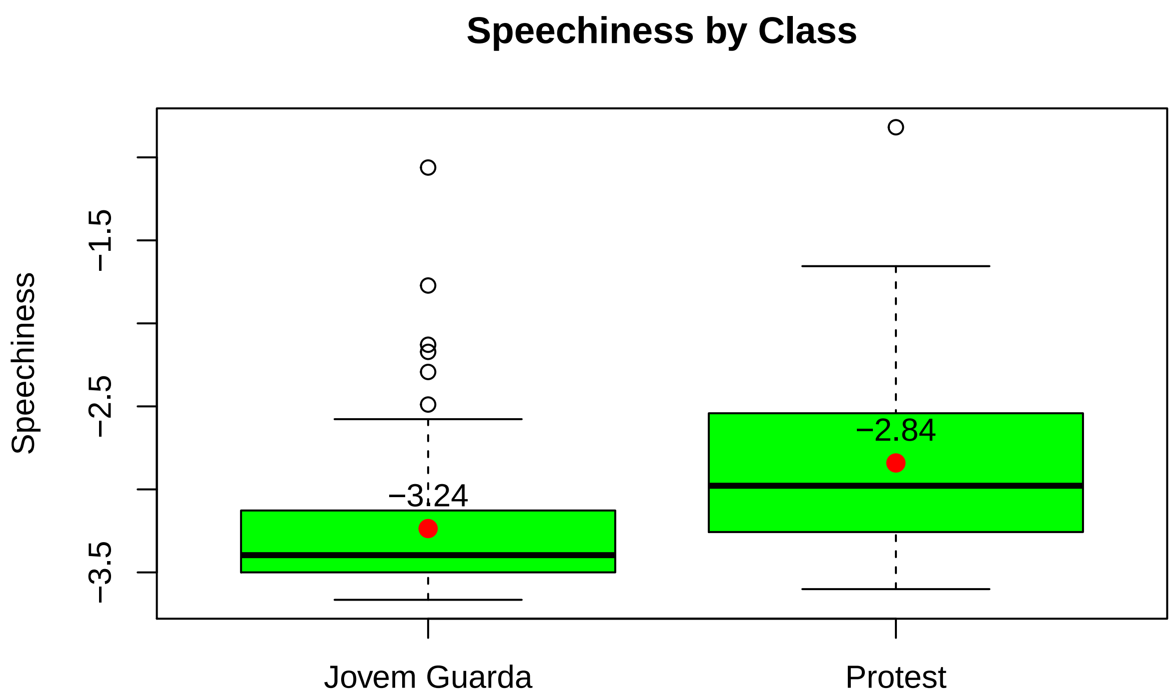
```
## The following object is masked from 'package:NLP':
##
##     annotate
```


Valence by Class
Valence
1.0
0.8
0.6
0.4
0.2
0.72
0.61
Jovem Guarda
Protest


Speechiness by Class
Speechiness
0.4
0.3
0.2
0.1
0.04
0.07
Jovem Guarda
Protest


Energy by Class
Energy
0.2
0.4
0.6
0.8
0.58
0.52
Jovem Guarda
Protest


Danceability by Class
Danceability
0.2
0.4
0.6
0.8
0.57
0.52
Jovem Guarda
Protest



From the boxplots above, it seems that there is visual evidence for a significant differences between the two classes of music (Jovem Guarda and Protest). Let's conduct some t-tests to evaluate if these differences are significant.

```
## [1] 0.05272099 0.17973901
## attr(,"conf.level")
## [1] 0.95
```

```
## [1] -0.03775548 -0.01186652
## attr(,"conf.level")
## [1] 0.95
```

```
## [1] 0.01013045 0.11459155
## attr(,"conf.level")
## [1] 0.95
```

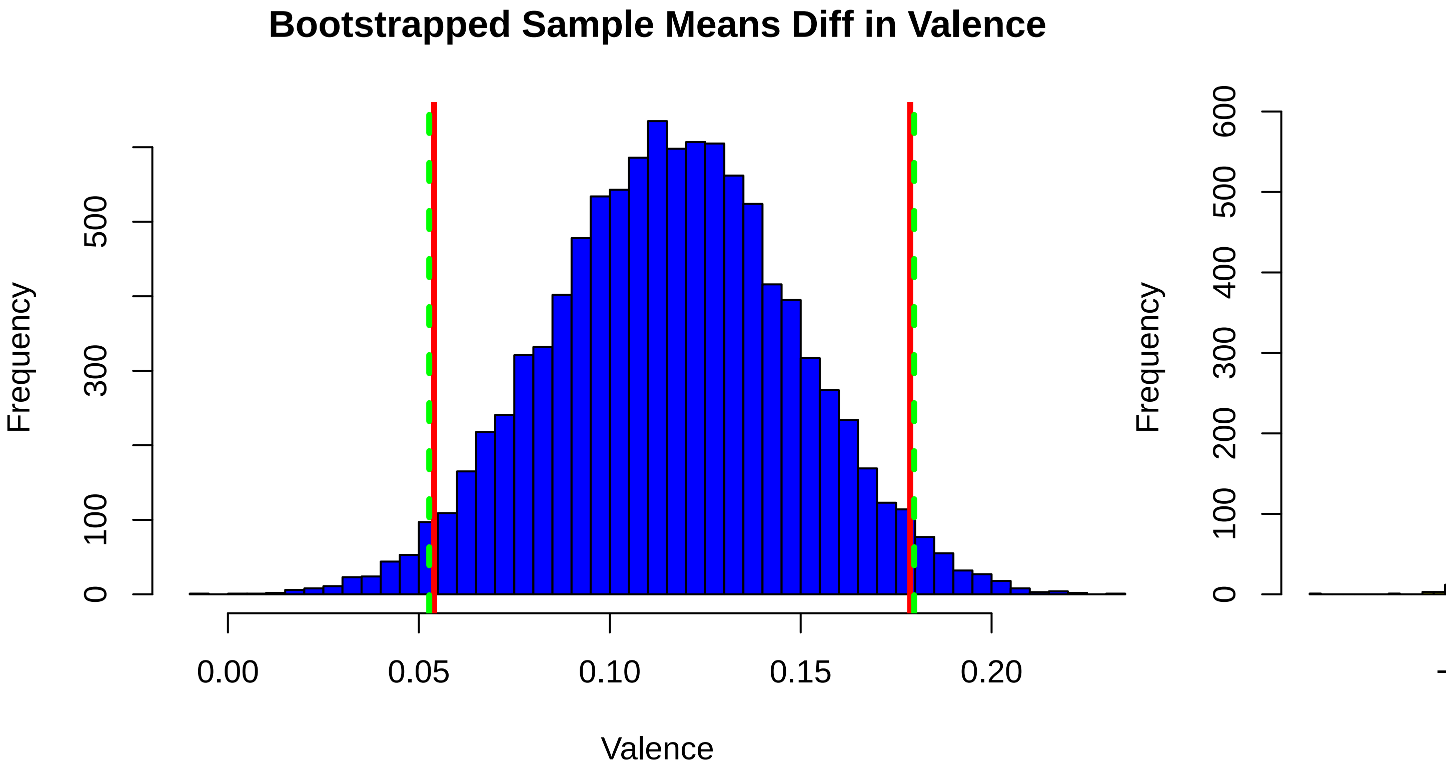
```
## [1] 0.009640185 0.077999815
## attr(,"conf.level")
## [1] 0.95
```

Permutation Test

Given that our t-tests p-values are significant, let's conduct a bootstrap test on Valence to look for the confidence intervals for the means difference in valence between the two classes.

```
N <- 10000
diffValence <- rep(NA, N)
set.seed(1)     #This is so we get same results every time

for (i in 1:N) {
  sA <- sample(music$valence[music$class == "Protest"],
              sum(music$class == "Protest"), replace = TRUE)
  sB <- sample(music$valence[music$class == "Jovem Guarda"],
              sum(music$class == "Jovem Guarda"), replace = TRUE)
```

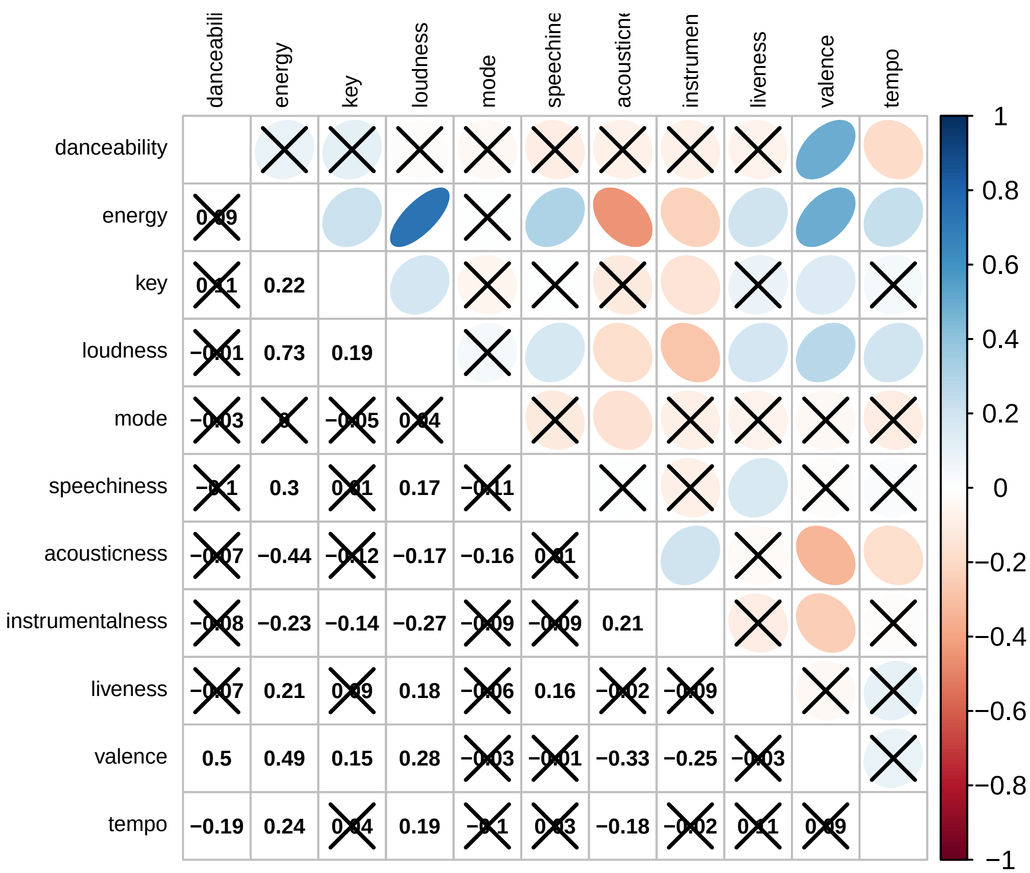
```
  diffValence[i] <-  mean(sB) - mean(sA)
}

boot_ci <- quantile(diffValence, c(0.025, 0.975))

#Make histogram of bootstrap sample means
hist(diffValence, col = "blue", main = "Bootstrapped Sample Means Diff in Valence", xlab = "Valence", b

#Add lines to histogram for CI's
abline(v=boot_ci,lwd=3, col="red")
abline(v=test1,lwd=3, col="green", lty = 2)
legend(48,600, c("Original CI","Boot CI"), lwd=3, col = c("green","red"), lty = c(2,1))
```



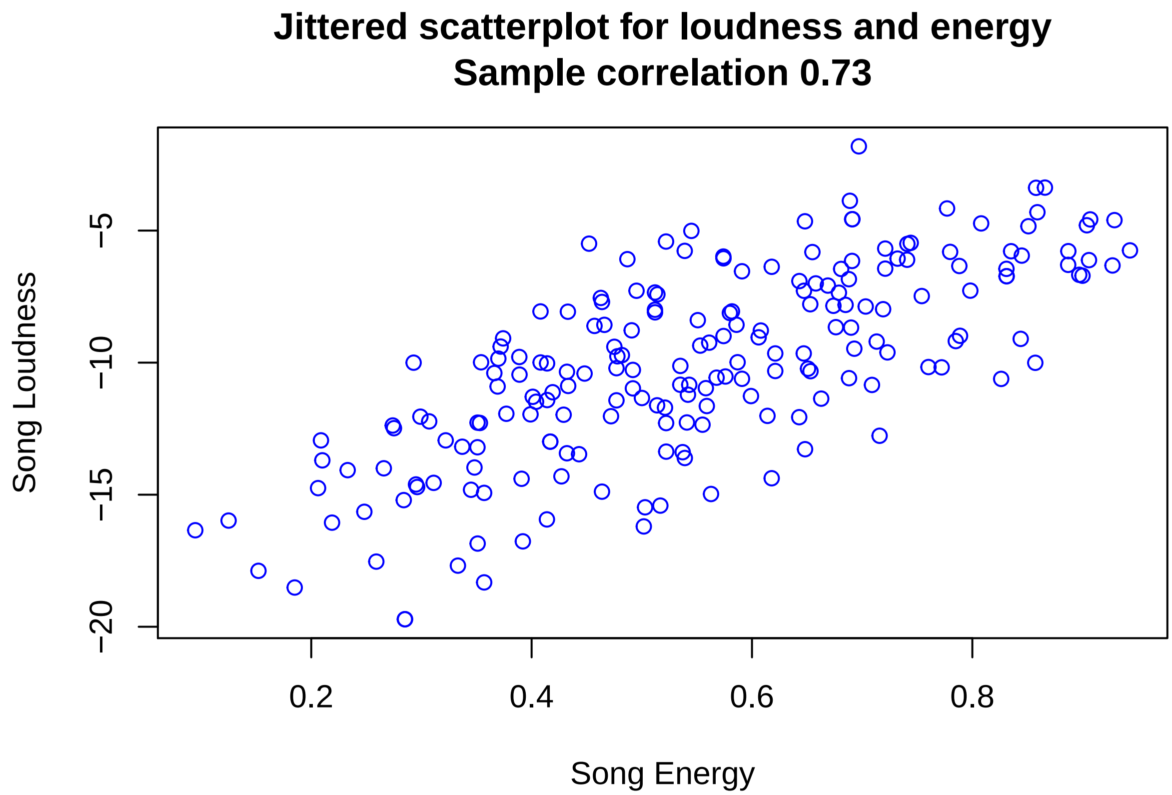
Basic tests with the different classes

```
## corrplot 0.84 loaded

##                  danceability energy   key loudness  mode speechiness
## danceability             1.00   0.09  0.11    -0.01 -0.03       -0.10
## energy                   0.09   1.00  0.22     0.73  0.00        0.30
## key                      0.11   0.22  1.00     0.19 -0.05        0.01
## loudness                -0.01   0.73  0.19     1.00  0.04        0.17
## mode                    -0.03   0.00 -0.05     0.04  1.00       -0.11
## speechiness             -0.10   0.30  0.01     0.17 -0.11        1.00
## acousticness            -0.07  -0.44 -0.12    -0.17 -0.16        0.01
## instrumentalness        -0.08  -0.23 -0.14    -0.27 -0.09       -0.09
## liveness                -0.07   0.21  0.09     0.18 -0.06        0.16
## valence                  0.50   0.49  0.15     0.28 -0.03       -0.01
## tempo                   -0.19   0.24  0.04     0.19 -0.10        0.03
```

```
##                    acousticness instrumentalness liveness valence tempo
## danceability              -0.07            -0.08    -0.07    0.50 -0.19
## energy                    -0.44            -0.23     0.21    0.49  0.24
## key                       -0.12            -0.14     0.09    0.15  0.04
## loudness                  -0.17            -0.27     0.18    0.28  0.19
## mode                      -0.16            -0.09    -0.06   -0.03 -0.10
## speechiness                0.01            -0.09     0.16   -0.01  0.03
## acousticness               1.00             0.21    -0.02   -0.33 -0.18
## instrumentalness           0.21             1.00    -0.09   -0.25 -0.02
## liveness                  -0.02            -0.09     1.00   -0.03  0.11
## valence                   -0.33            -0.25    -0.03    1.00  0.09
## tempo                     -0.18            -0.02     0.11    0.09  1.00

## [1] "The two column names of the two variables with the highest correlation:"

## [1] "loudness" "energy"
```



Now let's examine more closely the correlation between the two variables with highest correlation.



By adding a small amount of random normally distributed noise, we can see observations and their densities more clearly, and now it looks like there is a strong correlation between the two song features (as demonstrated by the slighlty linear concentration in density).

Stepwise Regression

We are now going to proceed with performing stepwise regression. In particular, we're going to fit a model that looks at possible predictors of the class of the song. To do this, I'm making a new dataset called `music2` which contains the relevant columns (notice I'm putting the response variable FIRST). Be sure to remove the option `eval = F`.

```
#avoid multicolinaeartiy issues
music2 <- music[,c(5, 8:18)]

names(music2)
```

```
##  [1] "song_popularity"  "danceability"     "energy"
##  [4] "key"              "loudness"         "mode"
##  [7] "speechiness"      "acousticness"     "instrumentalness"
## [10] "liveness"         "valence"          "tempo"
```

```
dim(music2)
```

```
## [1] 200  12
```

```
total_vars <- dim(music2)[2]
```

Perform best subsets regression using the `regsubsets` function in the `leaps` package. Save the results in an object called `mod2`. Get the summary of `mod2` and save the results in an object called `mod2sum`. Display `mod2sum$which` to get a sense of which variables are included at each step of best subsets.

```r
library('leaps')

mod2 <- regsubsets(song_popularity ~ ., data=music2, nvmax=total_vars)
mod2sum <- summary(mod2)
mod2sum$which
```

Now, let's examine the best model according to highest r-squared, etc.

```r
modnum = which.max(mod2sum$rsq)

#Which variables are in model 12
names(music2)[mod2sum$which[modnum,]][-1]
```

```
##  [1] "danceability"     "energy"           "key"
##  [4] "loudness"         "mode"             "speechiness"
##  [7] "acousticness"     "instrumentalness" "liveness"
## [10] "valence"          "tempo"
```

```r
#Fit this model and show results
musictemp <- music2[,mod2sum$which[modnum,]]

summary(lm(song_popularity ~ .,data=musictemp))
```

```
##
## Call:
## lm(formula = song_popularity ~ ., data = musictemp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -34.770  -9.199   1.369  10.531  32.899
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       48.20764   13.68885   3.522 0.000538 ***
## danceability       7.72240   11.08497   0.697 0.486879
## energy             2.28472   11.61569   0.197 0.844281
## key                0.02144    0.35644   0.060 0.952097
## loudness           0.19516    0.49409   0.395 0.693295
## mode              -6.39425    2.56830  -2.490 0.013654 *
## speechiness      -36.29120   26.05897  -1.393 0.165369
## acousticness     -12.91296    5.16885  -2.498 0.013340 *
## instrumentalness -23.67305   12.17040  -1.945 0.053250 .
## liveness           1.53966    5.76663   0.267 0.789766
## valence          -15.67833    6.80853  -2.303 0.022388 *
## tempo             -0.02327    0.04651  -0.500 0.617425
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.88 on 188 degrees of freedom
## Multiple R-squared:  0.09774,    Adjusted R-squared:  0.04495
## F-statistic: 1.851 on 11 and 188 DF,  p-value: 0.04826
```

```r
modnum <- which.max(mod2sum$adjr2)

#Which variables are in model 12
names(music2)[mod2sum$which[modnum,]][-1]
```

```
## [1] "mode"             "speechiness"      "acousticness"
## [4] "instrumentalness" "valence"
```

```r
#Fit this model and show results
musictemp <- music2[,mod2sum$which[modnum,]]
summary(lm(song_popularity ~ .,data=musictemp))
```

```
##
## Call:
## lm(formula = song_popularity ~ ., data = musictemp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -33.391  -8.639   1.188  10.007  32.930
##
## Coefficients:
##                  Estimate Std. Error t value            Pr(>|t|)
## (Intercept)        46.770      5.416   8.635 0.00000000000000214 ***
## mode               -6.233      2.499  -2.494             0.01346 *
## speechiness       -32.152     23.503  -1.368             0.17290
## acousticness      -12.832      4.548  -2.822             0.00527 **
## instrumentalness  -25.443     11.605  -2.192             0.02954 *
## valence           -12.281      5.158  -2.381             0.01824 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.7 on 194 degrees of freedom
## Multiple R-squared:  0.09072,    Adjusted R-squared:  0.06729
## F-statistic: 3.871 on 5 and 194 DF,  p-value: 0.00229
```

BIC

```r
modnum = which.min(mod2sum$bic)

#Which variables are in model 12
names(music2)[mod2sum$which[modnum,]][-1]
```

```
## [1] "acousticness"
```

```r
#Fit this model and show results
musictemp <- music2[,mod2sum$which[modnum,]]

summary(lm(song_popularity ~ .,data=musictemp))
```

```
##
## Call:
## lm(formula = song_popularity ~ ., data = musictemp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -28.775 -11.039   1.713  10.730  35.189
##
## Coefficients:
##              Estimate Std. Error t value            Pr(>|t|)
## (Intercept)    30.538      2.141   14.26 <0.0000000000000002 ***
## acousticness   -9.624      4.297   -2.24              0.0262 *
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.09 on 198 degrees of freedom
## Multiple R-squared:  0.02471,    Adjusted R-squared:  0.01979
## F-statistic: 5.017 on 1 and 198 DF,  p-value: 0.02621
```

CP

```
(modCP <- min(c(1:length(mod2sum$cp))[mod2sum$cp < c(1:length(mod2sum$cp))+1]))
```

```
## [1] 3
```

```
#Which variables are in model 2
names(music2)[mod2sum$which[modCP,]][-1]
```

```
## [1] "mode"         "acousticness" "valence"
```

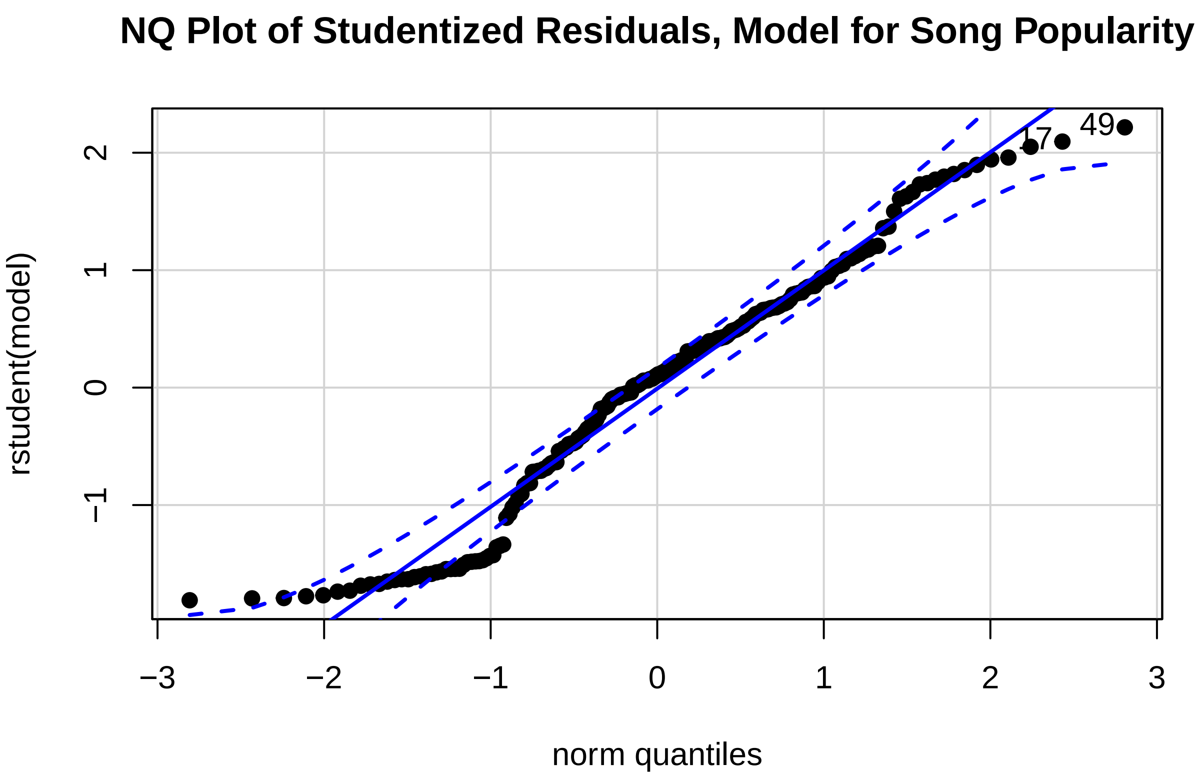
```
#Fit this model and show results
musictemp <- music2[,mod2sum$which[modCP,]]
summary(lm(song_popularity ~ .,data=musictemp))
```

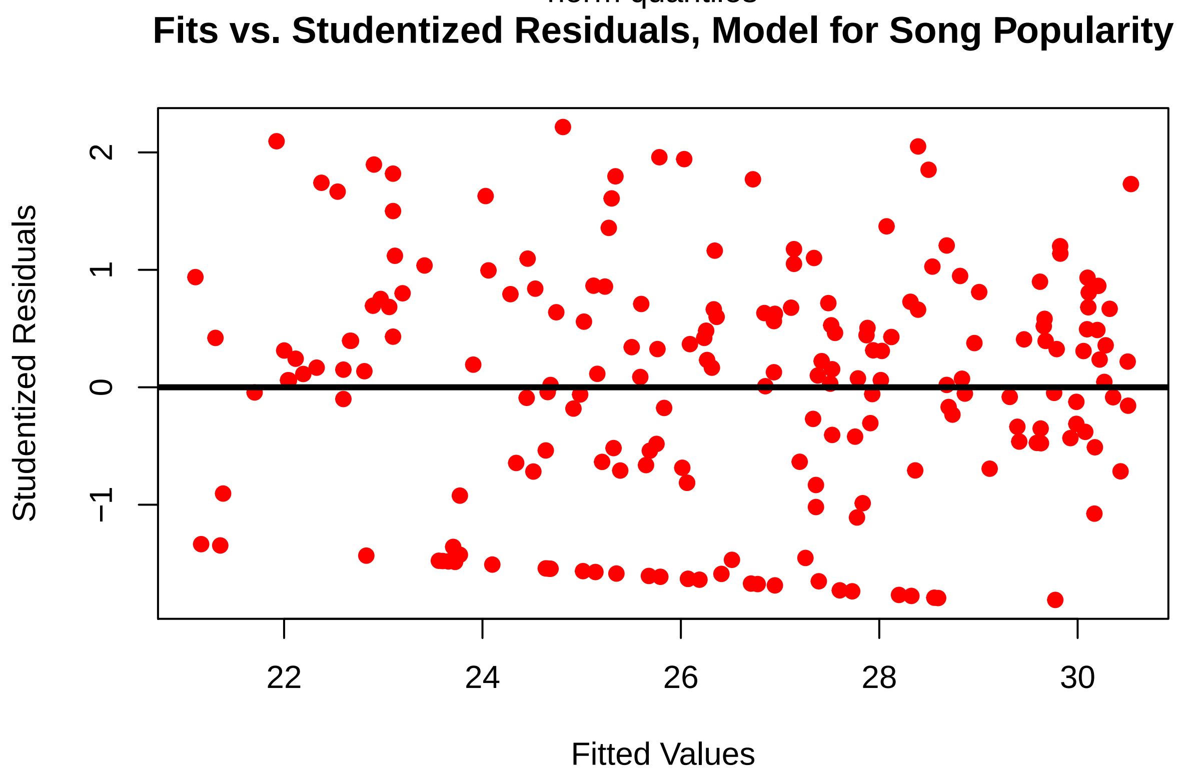
```
##
## Call:
## lm(formula = song_popularity ~ ., data = musictemp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -31.733  -8.423   1.979  10.724  32.436
##
## Coefficients:
##              Estimate Std. Error t value           Pr(>|t|)
## (Intercept)    42.783      5.129   8.341 0.0000000000000129 ***
## mode           -5.437      2.500  -2.175            0.03081 *
## acousticness  -13.974      4.560  -3.064            0.00249 **
## valence        -9.891      5.104  -1.938            0.05408 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 15.86 on 196 degrees of freedom
## Multiple R-squared:  0.062,  Adjusted R-squared:  0.04765
## F-statistic: 4.319 on 3 and 196 DF,  p-value: 0.005646
```

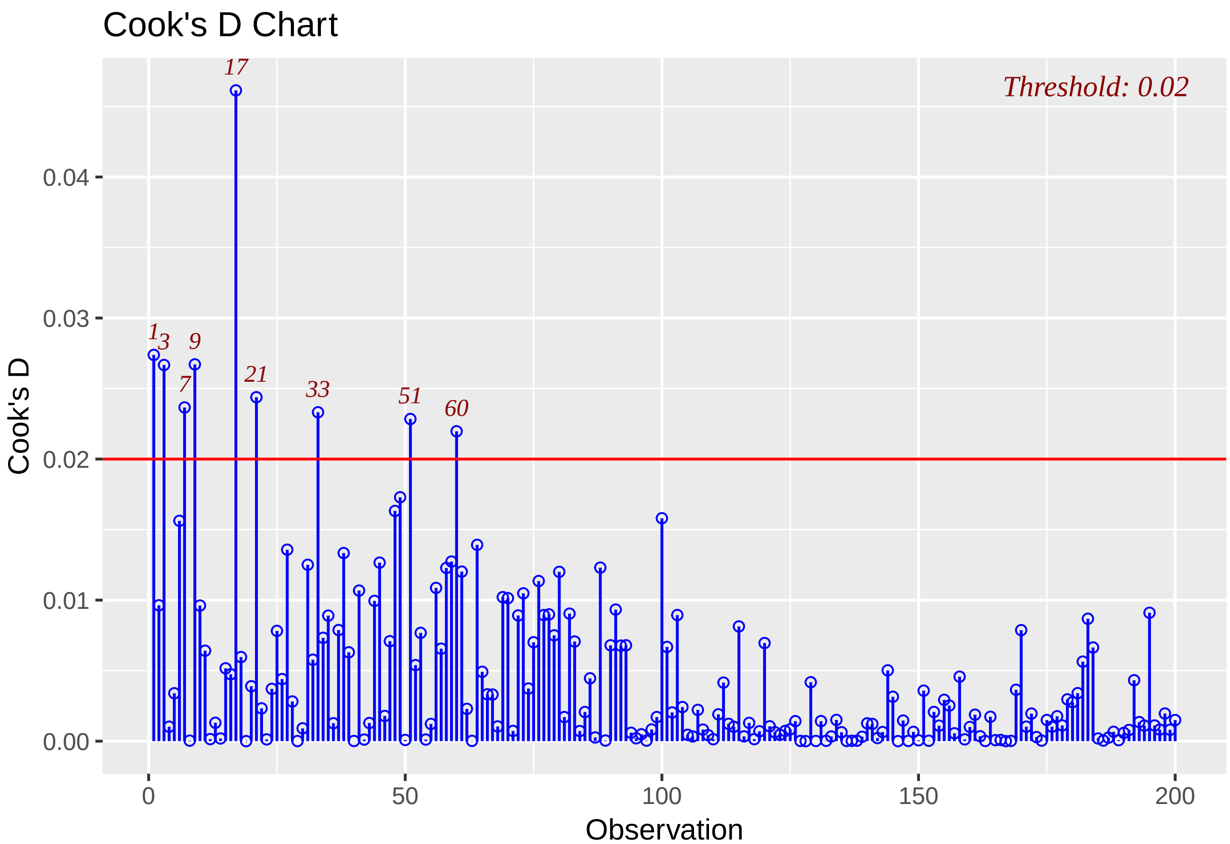
```
musicfinal <- music2[,mod2sum$which[1,]]
modfin <- lm(song_popularity ~ .,data=musicfinal)

#get new function for pairs plotn AND get myResPlots function
source("http://www.reuningscherer.net/s&ds230/Rfuncs/regJDRS.txt")
```

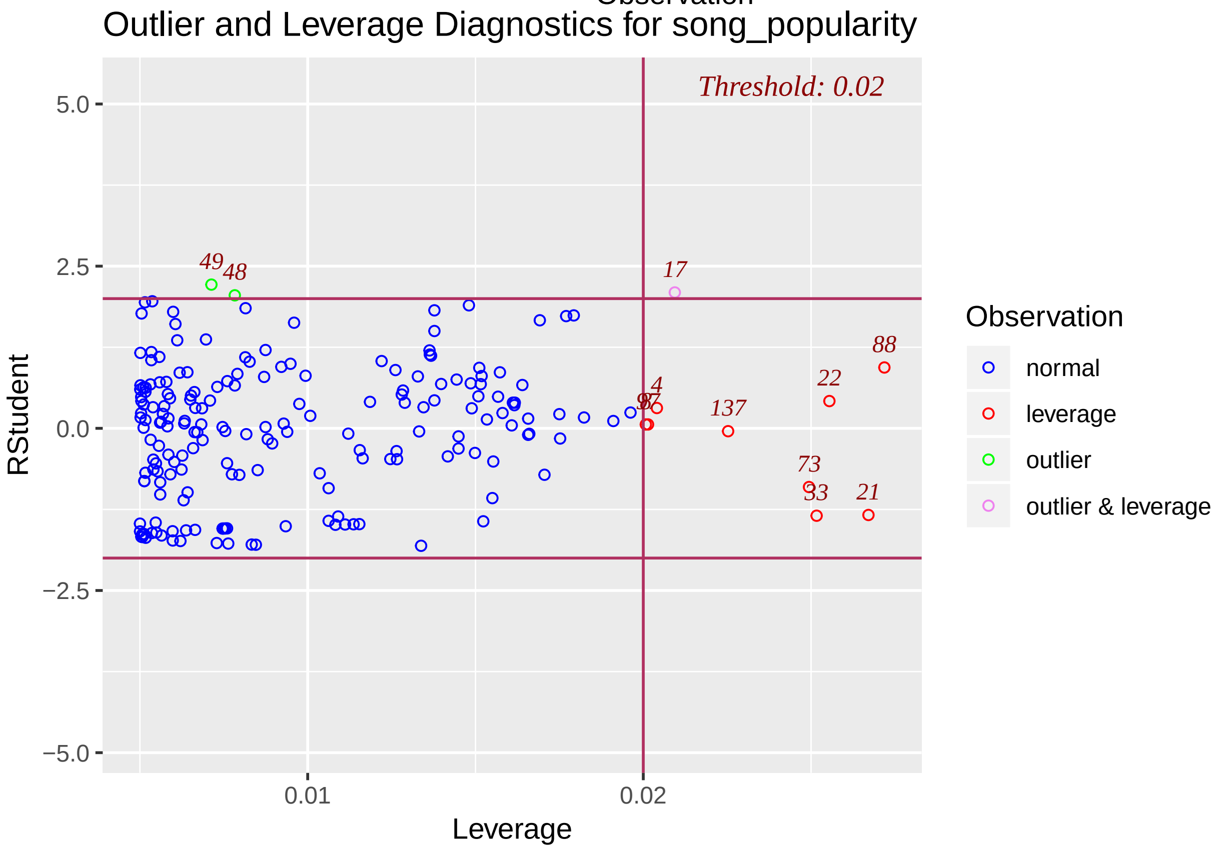
```
##
## Attaching package: 'olsrr'

## The following object is masked from 'package:datasets':
##
##     rivers

## Loading required package: carData
```

```
myResPlots(modfin,"Model for Song Popularity")
```


NQ Plot of Studentized Residuals, Model for Song Popularity
rstudent(model)
2
1
0
−1
−3
−2
−1
0
1
2
3
17
49
norm quantiles


Fits vs. Studentized Residuals, Model for Song Popularity
Studentized Residuals
2
1
0
−1
22
24
26
28
30
Fitted Values





Model seems tohave good fit, no evidence of heteroskedasticity,residuals seem to be approximately normallydistributed, although there is evidence that it is not normal distribution