Predict Album Genre by Title and Cover Image Attributes

John Dawes

November 1, 2019

Business Case

There are two scenarios that predicting an album genre by it's cover image attributes and title would be usefule

Scenario A: The Music Label

In the music industry, it is not unusual for labels to expect that a newly signed artist/band to have their all their branding ready to go for relase at contract signing. This includes the music being mixed and mastered, along with cover art for their first release on that label.

Scenario B: The Music Artist Looking to Get Signed on a Label

It would be useful for the artist to have a tool that would give them an idea of how much "like" their album cover communicates the genre of their music before they shop lables. For labels, to release their artists' sophomore (and later) albums, it would be a handy tool to input the album cover to get an idea of how much it represents the genre by album cover attributes.

Data Gathering

Some useful attributes of albums to predict genre would be the `title`, how many `faces` are on the cover, the most prevelant `colors`, identification of various `objects` and how abtract the cover image is - i.e. is it an `illustration` or a `photograph`. Thus the data set `albumData` below:

To gather these variables, I scraped several sources using a custom Node.js API that saved raw data responses into a MongoDB for gathering into well-formatted CSV as follows:

- *Artist Names* come from my personal music collection and is based on the folder names on my NAS' shared **Music** folder.
- For each artist name, a text search was performed via the Disccogs.com search API. This yielded up to the first 50 albums in an artist discography, that each contained: `year`, `title`, `genre`, `format`, and the all important `cover_image` url.
- Once all the album information from my artist collection was scraped, three Google Vision APIs were used to find how many faces, what color content, if there were any musical objects, and whether or not the cover is an abstract illustration or photograph.

Additional criteria for scraping artist albums were that they must be:

- The initial studio release (no singles, EPs, Live/Tour, Greatest Hits/Best of, Soundtracks, Remixes, Remasters, etc.)
- A CD or LP (to avoid duplicate entries from Tapes, file donwloads, etc.)
- Released in the United States (to avoid duplicate entries)

The following three sections describe how image attributes were aggregated from the Google Vision API image search results gather more variables for the album image covers:

Faces

The `faceDetection` function of Google Vision returns an object for each face found in an image, where those faces are (via coordinates for the eyebrows, eyes, and nose), and the probability of the emotion detected. Many album covers contain portraits of either a solo artist, the entiere band, or a human model. While being able to differentiate what kind of portraits/human subjects are in the photo would be useful, the album data only includes the number of faces detected in the image, for the sake of simplifing the use of `faces` as a predictor in `albumData`.

Colors

The `imageProperties` function of Google Vision returns an object that provides a percentage of each color found in the image. Each color is also scored as a percentage of the amount found in the image, and this is the value that is used to determined the most recurrent colors found in an album cover.

The colors values returned by `imageProperties` are RGB, and coverted to six-digit Hexidecimal via the custom Node.js API. However, 16 Million colors can be represented with the RGB and Hexidecimal values. It made more sense to convert this to a more managable list of factored values by converting the numerical color values into Color Names - i.e. from #FF0000 to "Red."

I found the `coloraze` npm module, which converts a Hexidecimal value to the closest color in it's pallette of 2196 color names. While this library is a massive reduction from the 16 Million colors that can be represented in Hex, I reduced the palete further to *16 web-safe colors* (See Figure 1: 16-Color Palette.) This was effective in distilling the essence of an album image's colors into a simplified color scheme and coverted the color values into factored predictors which should be enough to capture variance in album colors.

Figure 1: 16-Color Palette:

The list of colors used to reduce the range of 6-digit Hexidecimal colors from 16 Million continous values to 16 unique factors.

Captions

TBD - `labelDetection`

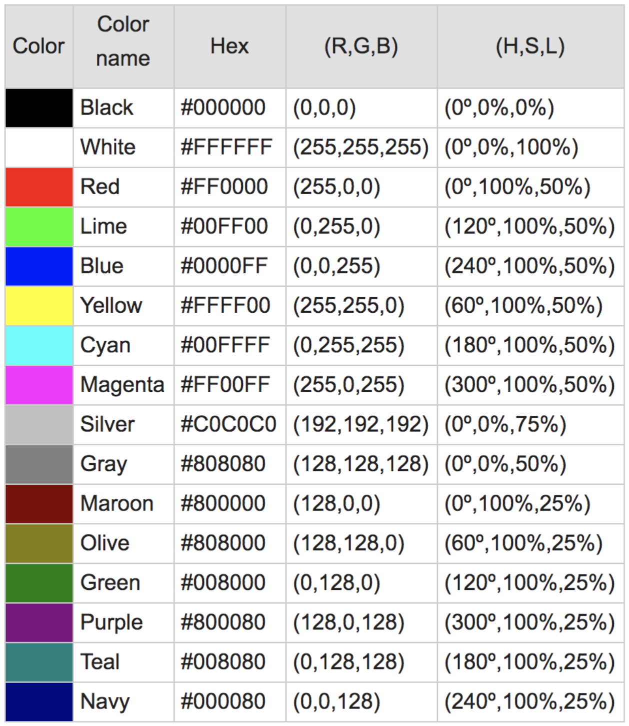
EDA

With the data scraping and aggregation completed, let's look at the raw data collected:

```r
albumCovers <- read_csv('data/album-cover-data.csv')

glimpse(albumCovers)
```

```
## Observations: 865
## Variables: 11
## $ artist          <chr> "Bauhaus", "Bonobo", "Delorean", "Haxan Cloak,...
## $ year            <dbl> 2005, 2013, 2010, 2013, 1978, 1997, 2017, 2009...
## $ title           <chr> "New York City, NY 11.12.05", "The North Borde...
## $ genre           <chr> "Electronic", "Electronic", "Rock", "Electroni...
## $ faces           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0...
## $ primary_color   <chr> "White", "Gray", "White", "Black", "Black", "O...
## $ primary_score   <dbl> 0.79221308, 0.52558565, 0.53635633, 0.40122879...
## $ secondary_color <chr> "Silver", "Teal", "Silver", "Gray", "White", "...
```



Color	Color name	Hex	(R,G,B)	(H,S,L)
⬛	Black	#000000	(0,0,0)	(0°,0%,0%)
⬜	White	#FFFFFF	(255,255,255)	(0°,0%,100%)
🟥	Red	#FF0000	(255,0,0)	(0°,100%,50%)
🟩	Lime	#00FF00	(0,255,0)	(120°,100%,50%)
🟦	Blue	#0000FF	(0,0,255)	(240°,100%,50%)
🟨	Yellow	#FFFF00	(255,255,0)	(60°,100%,50%)
🟦	Cyan	#00FFFF	(0,255,255)	(180°,100%,50%)
🟪	Magenta	#FF00FF	(255,0,255)	(300°,100%,50%)
⬜	Silver	#C0C0C0	(192,192,192)	(0°,0%,75%)
⬛	Gray	#808080	(128,128,128)	(0°,0%,50%)
🟫	Maroon	#800000	(128,0,0)	(0°,100%,25%)
🟫	Olive	#808000	(128,128,0)	(60°,100%,25%)
🟩	Green	#008000	(0,128,0)	(120°,100%,25%)
🟪	Purple	#800080	(128,0,128)	(300°,100%,25%)
🟦	Teal	#008080	(0,128,128)	(180°,100%,25%)
🟦	Navy	#000080	(0,0,128)	(240°,100%,25%)

Figure 1: Source: https://www.rapidtables.com/web/color/color-wheel.html

```
## $ secondary_score <dbl> 0.08269589, 0.24687445, 0.27873254, 0.29843235...
## $ tertiary_color  <chr> "Black", "Gray", "Silver", "Black", "Silver", ...
## $ tertiary_score  <dbl> 0.05008276, 0.13973045, 0.13341449, 0.29201144...
```

The 11 variables in `albumData` are as follows: * `artist` is the name of the music artist that released the album * `year` is the studio release * `title` is the title of the artist's studio release * `genre` which is our target classifier. * `faces` are the number of faces found by Google Vision in the album cover * The *3 colors with the highest scores* are saved in `albumData` as: `primary_color`, `secondary_color`, and `tertiary_color`. Their scores are also saved in the similarly named: `primary_score`, `secondary_score`, and `tertiary_score` for the potential of building interactive terms in linear and logistical models if need be.

```r
colSums(is.na(albumCovers))
```

```
##          artist            year           title           genre 
##               0              31               0               0 
##           faces   primary_color   primary_score secondary_color 
##               0               0               0               0 
## secondary_score  tertiary_color  tertiary_score 
##               0               1               1 
```

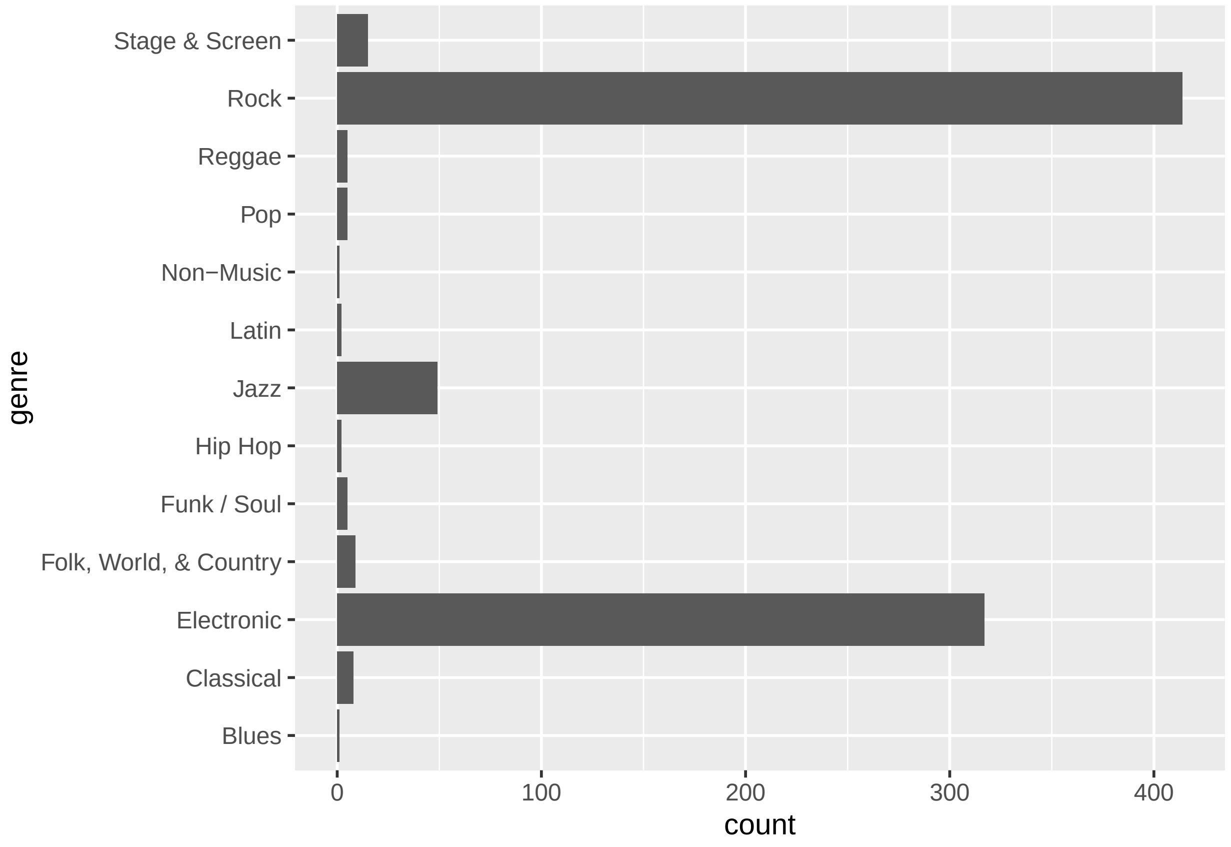
```r
clean_dat <- albumCovers %>%
  drop_na()
colSums(is.na(clean_dat))
```

```
##          artist            year           title           genre 
##               0               0               0               0 
##           faces   primary_color   primary_score secondary_color 
##               0               0               0               0 
## secondary_score  tertiary_color  tertiary_score 
##               0               0               0 
```

There are only 31 artists with missing album `years` as well as 1 record that didn't have a 3rd color. I've elected to remove these rows of data as I feel year will not be a good predicter in determining genre from the album image, which leaves the last observation with a missing 3rd color as insignificant.

Let's take a look and see how this collection of artists is distributed across years, and genres.

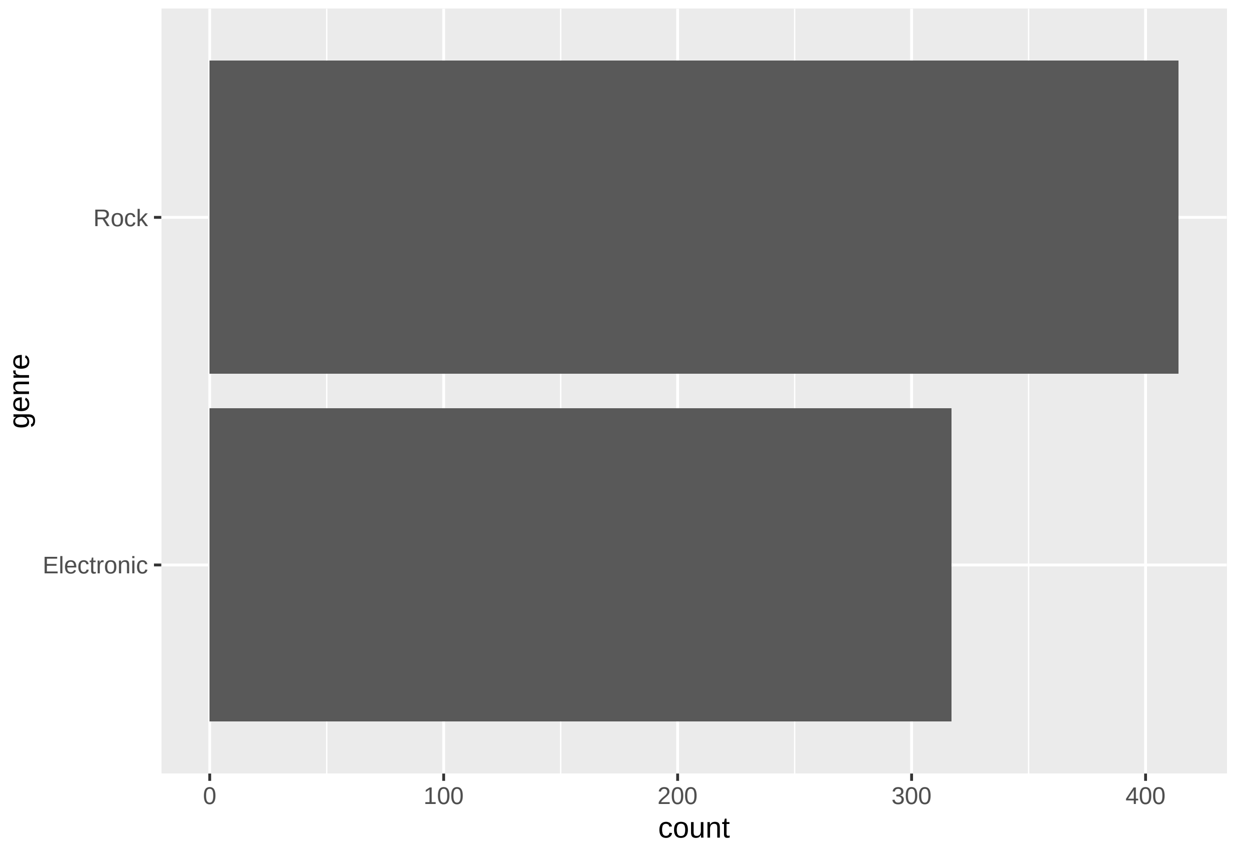
```
clean_dat %>%
  ggplot() +
  geom_bar(aes(x=genre)) +
  coord_flip()
```



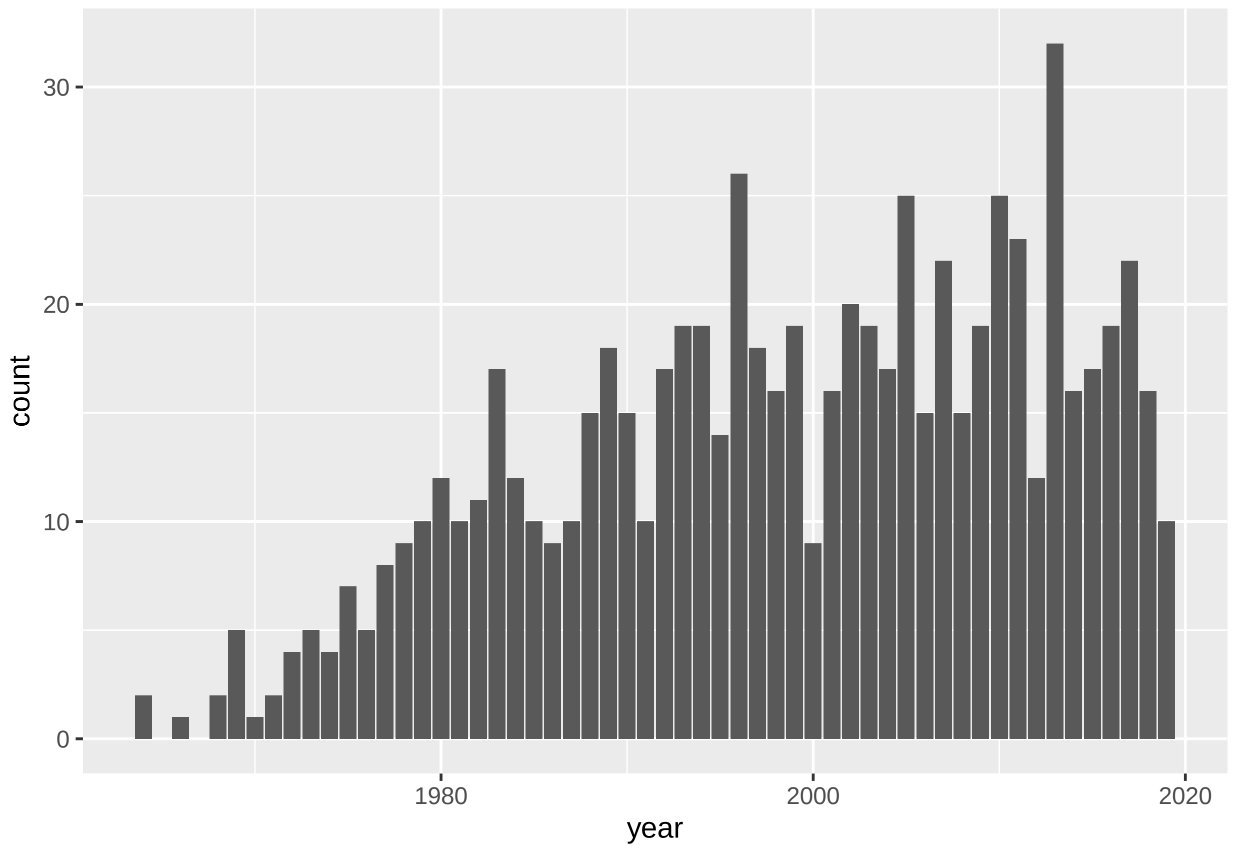
I'm definitely a huge fan of Rock and Electronic music. Not a significant amount of artist releases in the other genres! It also appears that some sound track ablums slipped through the scraping filters as `Sound & Stage`. The model would not get enough observations to anything but `Rock` and `Electronic`...so removing all the other genres would be helpful.

```
clean_dat <- filter(clean_dat, genre == 'Rock' | genre == 'Electronic')
```

```
clean_dat %>%
  ggplot() +
  geom_bar(aes(x=genre)) +
  coord_flip()
```

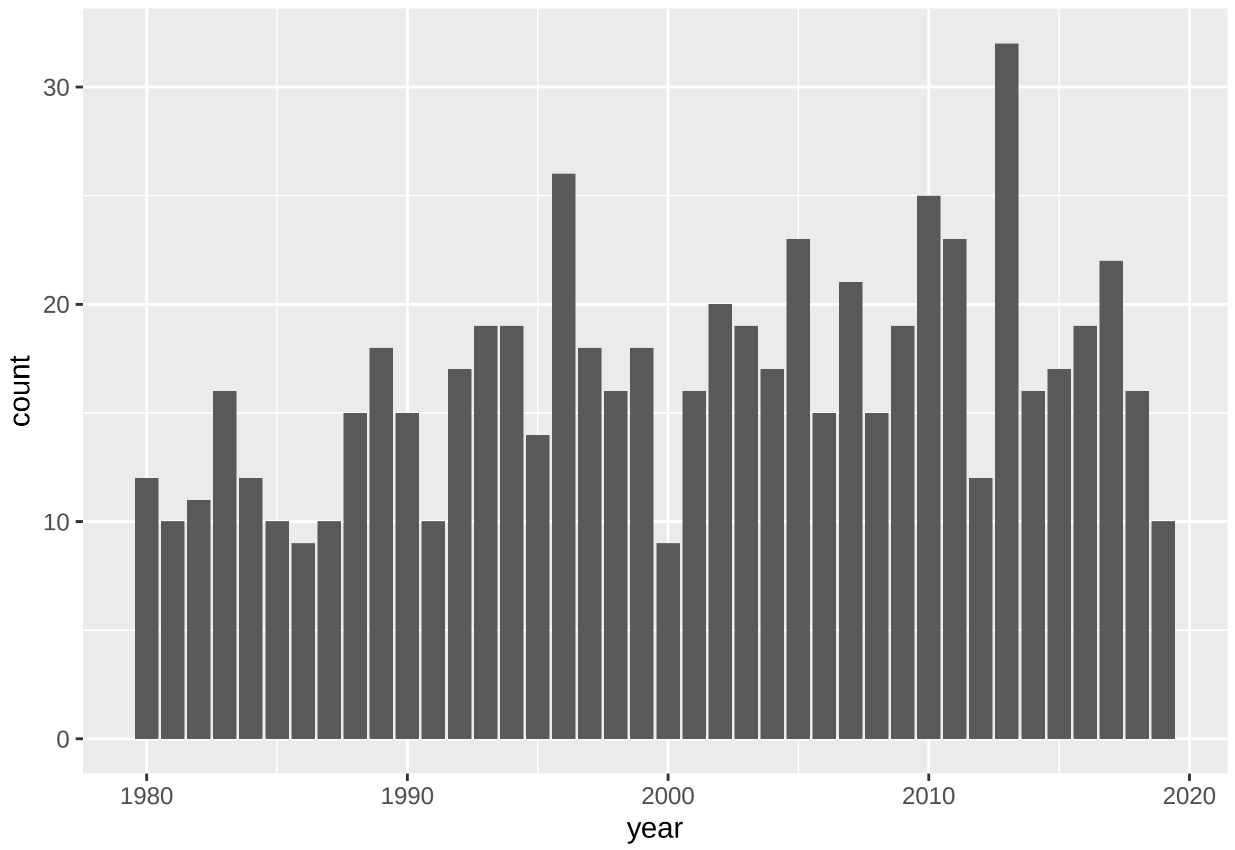


```
clean_dat %>%
  ggplot() +
  geom_bar(aes(x=year))
```



I'm definitely a child of the 80's and the number of albums don't pick up until then. For a better distribution, I've elected to filter anything older than 1980.

```
clean_dat <- filter(clean_dat, year >= 1980)
clean_dat <- filter(clean_dat, !str_detect(title, 'Tour|Live')) # Remove more non-studio releases
clean_dat <- filter(clean_dat, !str_detect(title, 'New York')) # Remove other live albums
clean_dat %>%
  ggplot() +
  geom_bar(aes(x=year))
```



```
num_raw_albums <- albumCovers %>% nrow()

num_raw_artists <- albumCovers %>%
  group_by(artist) %>%
  count() %>%
  nrow()

albumCovers %>%
  group_by(artist) %>%
  count() %>%
  ungroup() %>%
  summarize(
    avg_releases = mean(n),
    num_albums = num_raw_albums,
    num_artists = num_raw_artists
)
```

```
## # A tibble: 1 x 3
##   avg_releases num_albums num_artists
##          <dbl>      <int>       <int>
## 1         3.37        865         257
```

```
num_albums <- clean_dat %>% nrow()

num_artists <- clean_dat %>%
  group_by(artist) %>%
  count() %>%
  nrow()
```

```r
clean_dat %>%
  group_by(artist) %>%
  count() %>%
  ungroup() %>%
  summarize(
    avg_releases = mean(n),
    num_albums = num_albums,
    num_artists = num_artists
)
```

```
## # A tibble: 1 x 3
##   avg_releases num_albums num_artists
##          <dbl>      <int>       <int>
## 1         2.86        661         231
```

After **NA's** are removed, there are `661` ablums from `231` artists, with an average of `2.8` releases each. In the original raw data there were `865` albums with just over `250` artists averaging `3.3` albums. Not much of a shift in the number of releases, which further shows that most of my music from 1980 onward is Rock and Pop.

Machine Learning

Supervised learning is an excellent canditate for training a machine on album attributes, as we already know the `genre`, we'll train a machine with linear and logistical regressions to see which is most effective in predicting the genre of an album. Interactive terms will also be used to emphasize the color related data variables to tease out more variance.

Analysis