Intuition versus statistics: Do user created playlists share statistical commonalities with algorithm created playlists of similar genre?

Jashan Chopra

University of Colorado Boulder

ASEN 5307 - Data Analysis Methods

I. Nomenclature

API = Application Programming Interface
SVD = Singular Value Decomposition
FLS = Flexible Linear Regression

II. Introduction

Throughout the semester we've used a variety of techniques to explore the significance of data. In particular, we've taken a look at heavily science orientated datasets, like sea level. However, data is increasingly become a crucial asset for businesses across the globe. One instance of strong usability for data aggregation and analysis is for music, particularly for applications like Spotify. Curated playlists created by Spotify's proprietary algorithms are used to generate interest in their premium paid service, as well as provide advertising to artists. Personally, I've always been an avid fan of creating playlists using Spotify, whether for my daily use or for special events. Occasionally I'll want a playlist for an outdoor barbeque, or a playlist that gathers some of my favorite new songs. Unfortunately, making playlists can be quite time consuming, but I listen to a lot of new music that I want to recall for later listening. One of my strategies for this has been to make a series of playlists that correspond to different moods, named by colors. For example: "yellow" is a playlist of upbeat songs that get me excited to start the day, whereas "black" is a playlist of moody hip-hop songs heavily laden with drums indicative of "trap-style" beats. I categorize songs into these playlists based on pure intuition, where I listen to the song and put it into these mood playlists based on how I feel after listening to it. This strategy has worked pretty well in my mind, but I am extremely curious to see how well Spotify's recommendation system and statistics agree with my intuition on songs. To do this, I'll be analyzing the Spotify API features, such as "danceability", between my playlists and Spotify algorithm generated playlists. Ideally I'd like to gain insight into how Spotify uses their aggregated data to create playlists, and compare it to my biased human intuition.

III. Literature Search

The literature on this subject can be sparse, mainly because most of the actual information of how Spotify's playlist development algorithms work is highly proprietary. Most public data on the Spotify algorithms attempt to take the public API features and apply them to basic machine learning models in an effort to figure out which markers best predict song popularity. Since we haven't explored any machine learning algorithms in class, it's difficult to utilize the methods in these papers directly. Pretty much all of the papers I could find focused on popularity, in some cases the papers analyzed popularity from a geographic lens. From a pure importance type of view, I think that looking at attempting to predict song popularity makes the most sense. However, I really wanted to have the personal perspective be a part of this data analysis project. So although I will look at the markers corresponding to popularity in my paper, the sources discussed below will mostly support my paper in understanding of the markers that the Spotify API reports for songs.

My first source is a paper I found published through an Italian University describing a model based approach towards data analysis of Spotify song data. I found it quite interesting that the research paper dived into the statistical management of song audio features before diving into an analysis on popularity. The second portion of the paper, which discusses an implemented model for determining popularity is similar, although far more advanced, to the second portion of my project. They also follow my reasoning for the importance, where identifying which audio features have a higher sway in the song popularity supports the success of new products [4].

My second major source is a paper by a master's student from the University of North Carolina. I think it's a solid paper because it follows a very similar course to the one that I proposed for my project, although it does also dive into the creation of a machine learning algorithm. This paper also includes some great discussion on the song features provided by the Spotify API. Furthermore, the student who wrote this paper does some principal component analysis, which I was super curious about because I was struggling to understand that section in the class [5].

My third source is similar to the last two, but is actually a published and peer reviewed research paper, which was one of the requirements for this project. This one also analyzes trends in music popularity using the API provided music features. Interestingly this source does some temporal analysis, something that my project will not look at. Their goal is more focused on trying to predict what future overall trends in music will be popular, instead of focusing on individual property of songs from singular characteristics [6].

My fourth and fifth sources provide more support for the basic data collection and analysis methods I'll be using to analyze Spotify data. The intrigue of these two sources was their side-focus on geographic data. I've decided not to do a geographic analysis and stick to the linear regression work, but I thought it was unique to consider whether some of this analysis would change with any geographic relation. It had me wondering if Spotify's algorithm actually includes geographic location and if so, how much do recommendations differ based on the user's location? I would assume so for language purposes, but not for a song's emotional characteristics [7][8].

My last source was one of the few papers I could find that utilizes a linear regression based model instead of a machine learning algorithim to predict popularity from song characteristics, which I found great because that is what I plan on doing based on what we've done in class. The paper uses a model we didn't cover called flexible least squares. Their FLS method predicts the number of weekly streams of song from those acoustic features. It will be good to compare the results of this source with my results. [9].

IV. Problem of Interest

A. Importance

The large majority of profit from some of the top internet businesses, such as Facebook or Google, comes from data oriented advertising. Some journalists now even claim that data is the most valuable resource globally, more expensive than crude oil [1]. Basically every software company that is successful today leverages statistics to improve aspects of their business. This especially holds true with Spotify, which collects a wide variety of data that supports their product. For this case, we will be discussing data that Spotify creates and collects for every song in their collection. Spotify uses these characteristics to classify songs, allowing them to be organized automatically into playlists based on any number of song characteristics, like genre, artist, energy, tempo, etc. Spotify as a music platform is extremely popular because it uses advanced models to recommend music to their users. If you are interested in a certain mood, or a specific genre, or popular new music, really whatever you can think of, then it's likely that Spotify has a pre-generated playlist for it. They also create weekly playlists of song recommendations for users, including daily mixes. None of these features, which lends them money in the form of user's premium subscriptions, would be possible without complicated data analysis techniques [3]. This project will personally give me insight into the Spotify API and show how useful some of these characteristics are, the knowledge of which can easily be applied to a wide variety of other subject matters. This skill of taking a large amount of seemingly unrelated data and using it to try to predict something like song popularity is an important skill to have working at many data oriented companies like Spotify.

B. Learning Goals

For this project I collected information on my own playlists, as well as a Spotify created playlist that I feel relates to the emotion that I picked when creating the playlist. For example, my blue playlist emotionally relates to a Spotify playlist called "Golden Hour." I want to know if I can use statistical analysis to find how well my intuition on creating playlists that match a certain emotion compares with playlists created by the Spotify algorithm. This learning goal is personal in nature, but also displays some of the importance of the Spotify algorithm. It takes time to create personal playlists, and if the generated playlists match many of the same characteristics as my own playlists than Spotify will be successful in keeping me around because I will listen to their playlists.

Looking into the API characteristics for these personal reasons will give me interesting insight into the overall capabilities of the Spotify algorithm. A reasonable follow up that relates more to the research papers I found in my literature search is to attempt creating a pedestrian version of their algorithm, using simple linear regression techniques that we've discussed in class. This is actually often the first step done in more complex machine learning examples. The examples typically always start with basic linear regression which then leads into more typical machine learning methods to better predict song popularity. Therefore the learning goal of this section is to see how well basic regression methods work with extraordinarily complex subject matter like song popularity, and to gain practice with the basic framework for more complicated machine learning projects.

C. Data Gathering

Since I wanted to use specific playlists in the analysis, I had to collect the data myself. I did this using the public Spotify API, which provides a ton of useful information and makes it easy to acquire Spotify data [2]. Accessing the API via Python is easy utilizing the Spotipy library. First, I had to create a developer account through Spotify and create an example application, where I gather my user credentials from. We then simply use the Spotipy library to create a user token from the credentials, and create a Spotify client object. With this Spotify client object I was able to access my playlists as well as Spotify public playlists using their unique playlist hash values.

For the data sorting, I turned the playlist hash from Spotify into a Pandas dataframe. Each row of the dataframe for a playlist includes the name of the song, the album, the artist, the unique track id, and then each of the API's audio features. Because connection to the Spotify API takes a decent amount of time, I've transferred this Pandas dataframe into a csv file for each playlist which I then load in my main function for ease of testing.

D. Data Analysis Plan

After reviewing the literature I found on the subject as well as my personal learning goals, I've decided on a two fold data analysis plan. The first focus of my data analysis is on comparing the two playlists. First I will create pairs of playlists for each emotional category, with one playlist being my playlist and the other being a similar Spotify playlist. The rest of the data analysis will be completed on each playlist pair in a loop. First, I will make a histogram of the API features, overlaying the two histograms for each playlist in each playlist set. Along with each histogram I will create a probability plot, and then do a first order visual analysis to question if the API features are normal for a given emotional category and compare the features between the two playlists in each set. I will also acquire the mode, median, and standard deviation, and min/max for each API statistic in this loop for potential later use. After this visual analysis, I will conduct a series of statistical tests to analyze the data. I will use a chi-squared test with a 95% confidence value to gauge the normality of the histograms, and then use a Student's T-test to see if the data from each playlist comes from the same population, also with a 95% confidence value. I will also do an f-test to compare the variance of each playlist, finding if the standard deviations are close and if the data is from a comparable origin, which I expect to align with the results of my Student's T-test. This will be the full data analysis for the comparison of playlists.

The second focus of the project will be centered around the API features. First, I will see if there are any correlations across all the features, for example I expect that "danceability" correlates fairly well with "energy". This will give me a better sense of these characteristics and see if there are any that I can essentially ignore off the bat. I will plan to use a SVD or QR decomposition for any linear regressions that appear to be singular in nature. After that, I will perform a multiple linear regression comparing each characteristic to the song popularity. I will test the regression coefficients with an f-test to see which characteristics are the best predictors of song popularity. If I do not have enough samples, which I somewhat anticipate since these playlists don't contain ridiculous amounts of songs, then I will bootstrap my

samples to attempt to create a better model. Once I've finalized my linear regression model I will analyze how well it works. I expect that the model will be poor, which will lead to a discussion of what additional techniques could be used and why machine learning is better for this type of data than basic regression techniques.

E. Analysis Results

Due to the number of figures, I've made them slightly smaller to fit 2 on the same page. I've included full resolution images under the /images/ folder in the final project zip file, if wanted.

As noted in the data analysis plan, each subplot of histograms below compares the API features across a pair of playlists. For example, 1 plots overlaying histograms across each API feature for the "Yellow" playlist, which is my custom playlist, compared with a Spotify created playlist of a similar mood: "My Life is a Movie." I've chosen to plot a subset of API features instead of all of them. I've removed "speechiness", as this feature simple detects if there are spoken words in the track. In my opinion, the mood of a playlist does not depend on whether or not their are vocals, thus for comparison I've removed this from the emotion analysis. Another feature, "instrumentalness", essentially acts as the same characteristic, detecting the presence of instruments in the music. I've also removed "acousticness", which defines if the track is mostly composed of acoustic instruments, again because I feel that this is not representative of the emotional characteristics of a song. Finally, I've removed "time_signature", "mode", and "key", because similar to the last features, these ones do not describe the emotions of a piece, but rather elements of musicality.

The remaining characteristics define the emotional context of a playlist. Features like "danceability", "loudness", "duration_s", and "tempo", and "popularity" are self-explanatory as words, however Spotify does not provide information as to what quantitative properties of a song are used to define these. Danceability includes elements of tempo, rythm stability, beat strength, and regularity [5]. Energy measures the intensity and overall activity of the song. The API suggests that tracks with a high energy score will feel fast, loud, and noisy, consisting of songs with high dynamic range, loudness, timbre, onset rate, and entropy. Liveness attempts to measure the probability that the track was performed live, which I decided to include because I do think that live versions of songs can sometimes deliver different feelings than studio recorded songs. Valence attempts to describe the musical positively of track. The API suggests that tracks with a high valence score are happy and cheerful [2]. First we will display these histogram subplots for each playlist pair, along with a probability subplot of features for each playlist across all the pairs. I will then discuss my first thoughts from visual analysis.

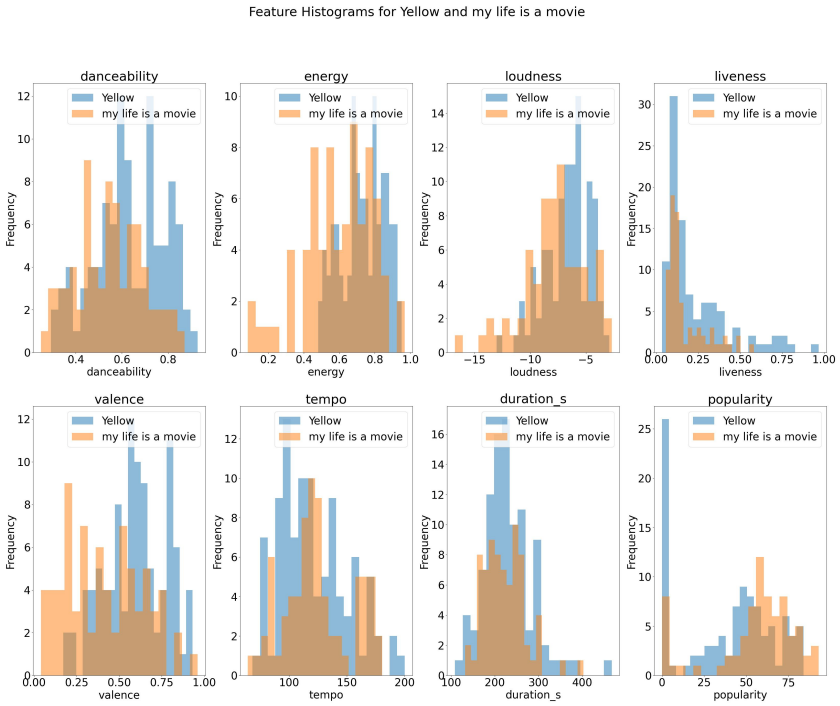


Fig. 1 Playlist "Yellow" compared with "My Life is a Movie"

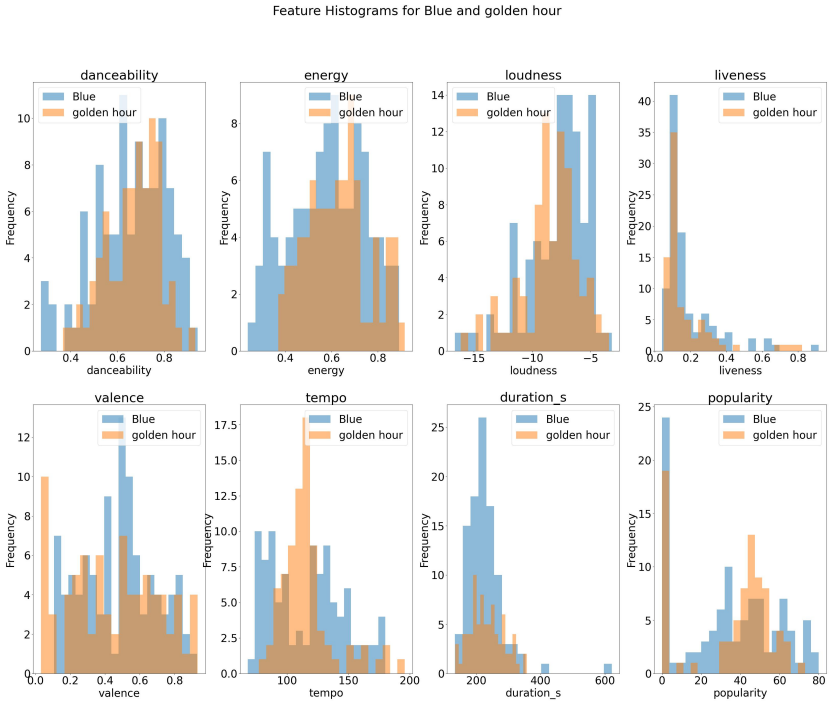


Fig. 2 Playlist "Blue" compared with "Golden Hour"

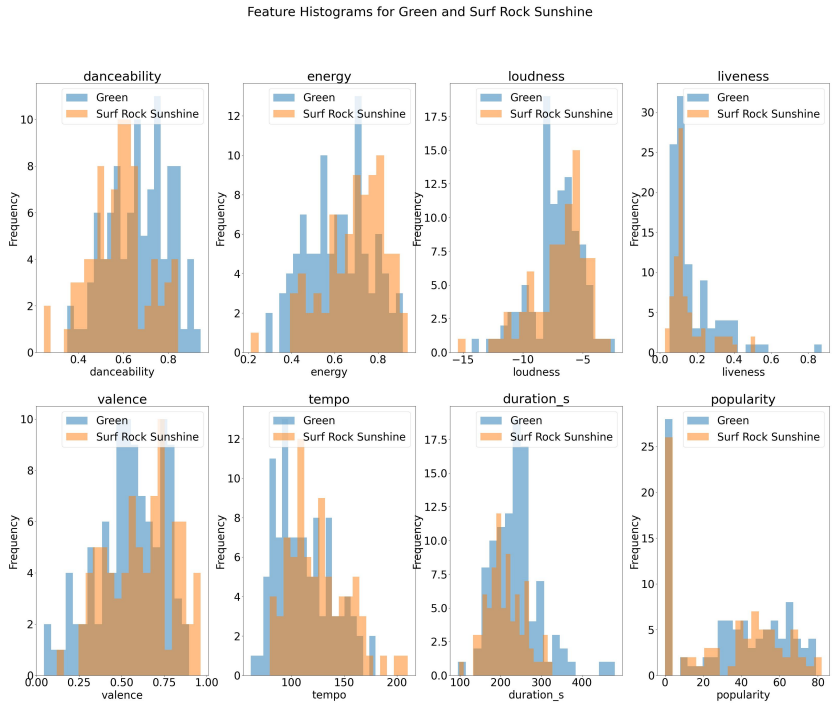


Fig. 3 Playlist "Green" compared with "Surf Rock Sunshine"

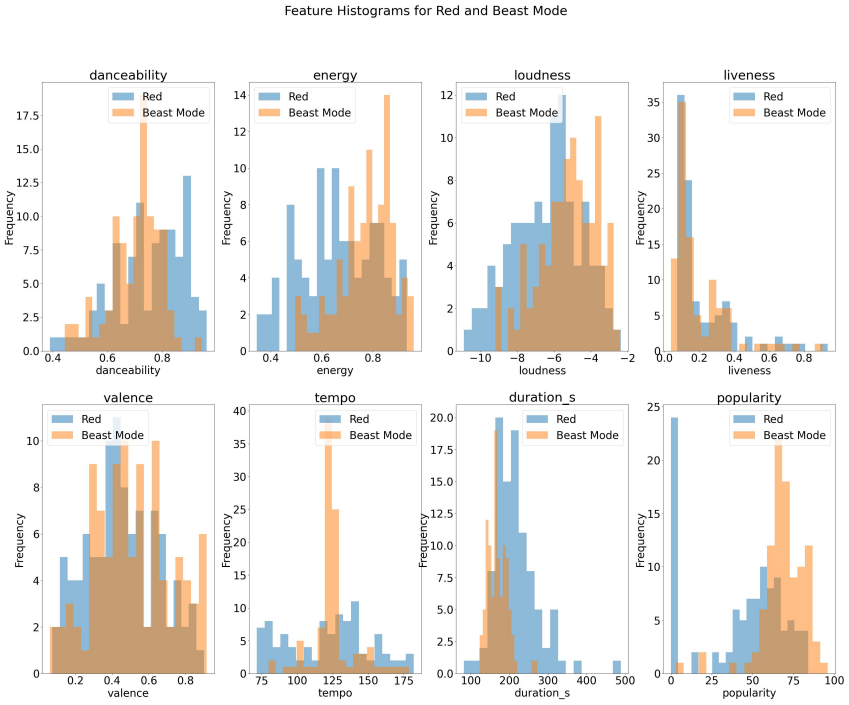


Fig. 4 Playlist "Red" compared with "Beast Mode"

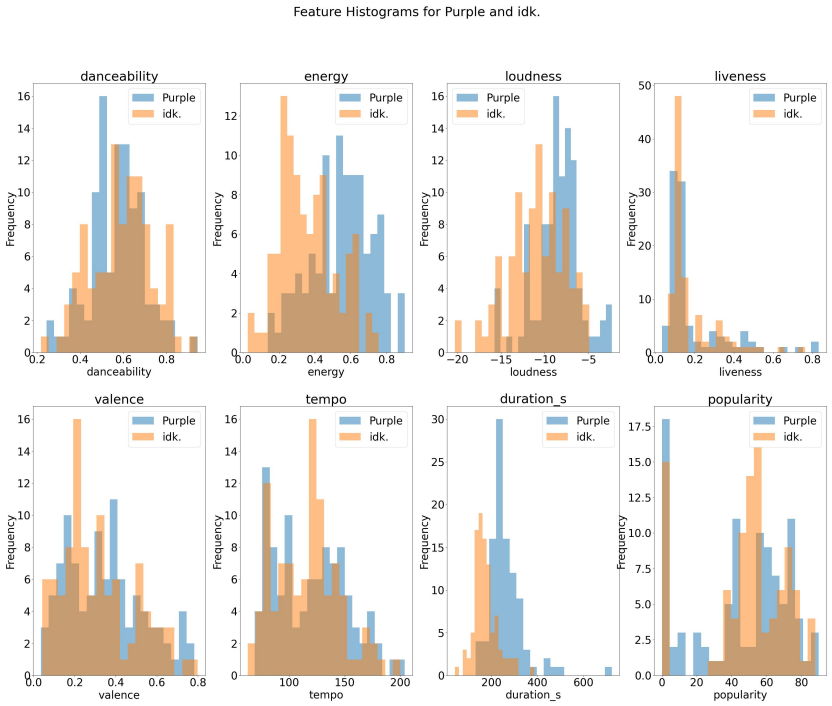


Fig. 5 Playlist "Purple" compared with "Idk"

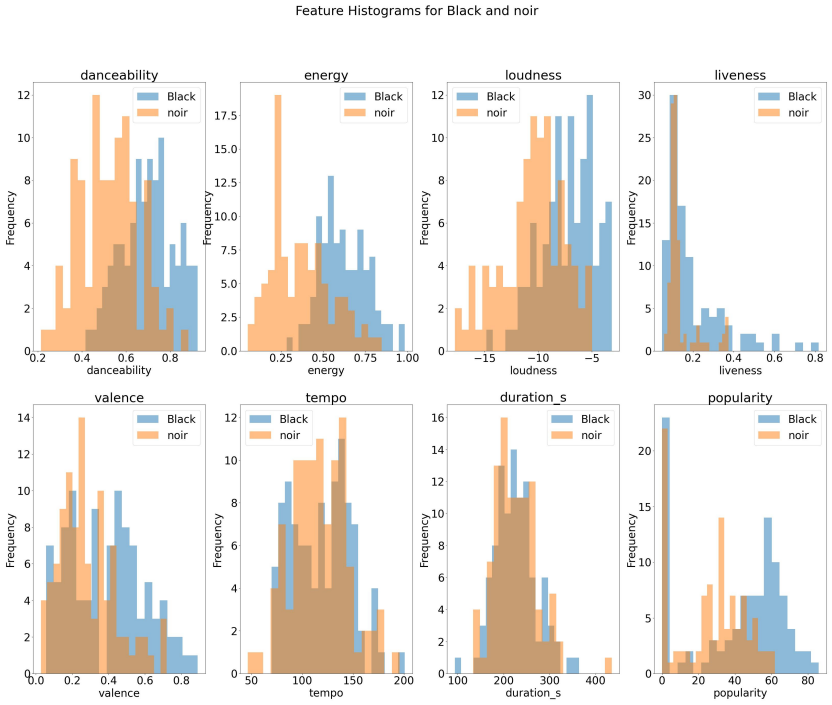
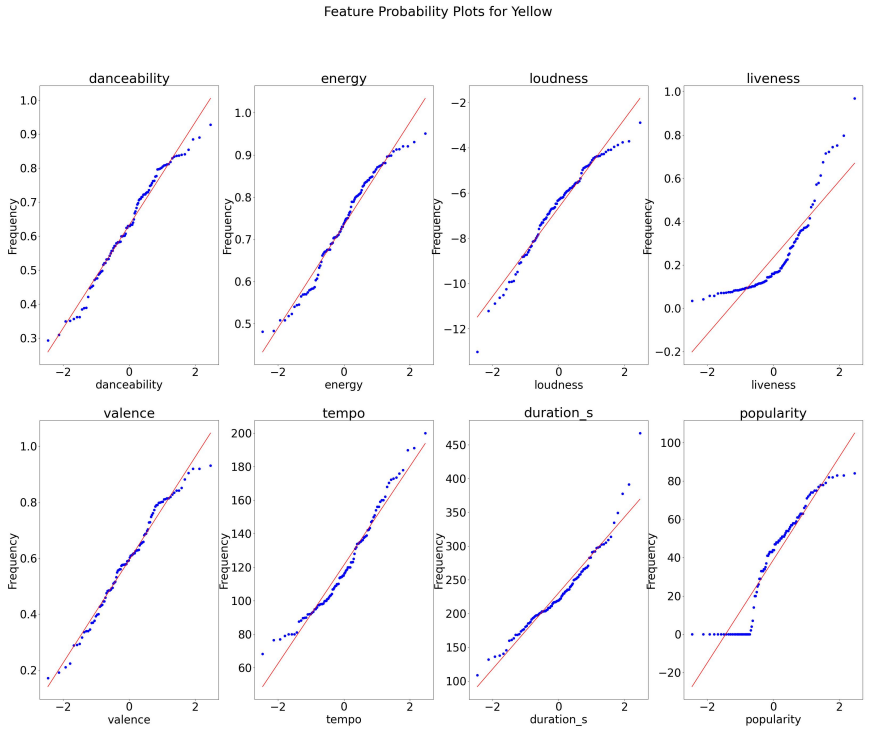


Fig. 6 Playlist "Black" compared with "Noir"

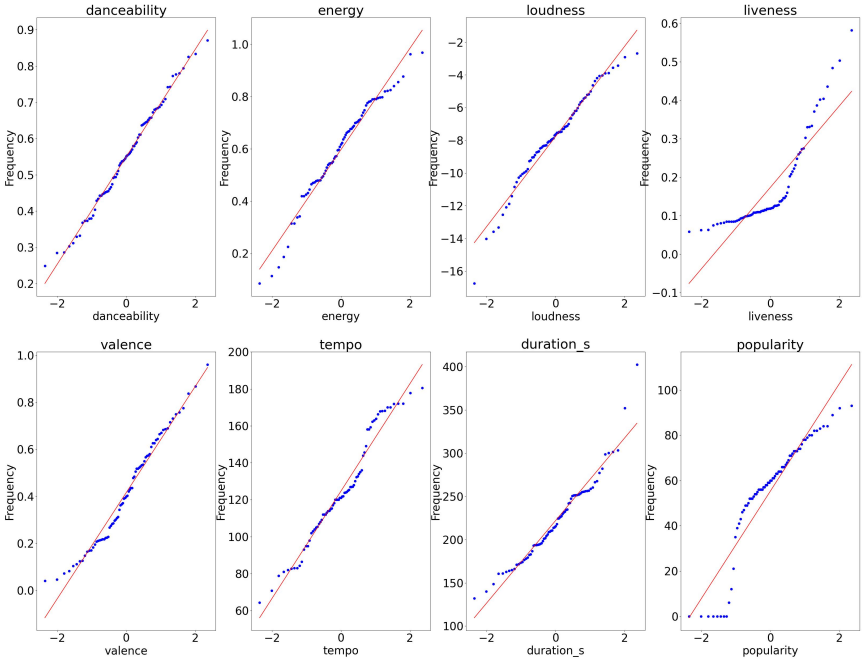
The first thing that I noted was that my playlists appear to be decently similar across the board, whereas the feature histograms for the Spotify AI generated playlists show stark differences. As an example, let's look at the valence feature. The "Golden Hour" playlist has a large spike a low valences. This makes sense, often relaxing songs can have sad undertones. Similarly, the "Noir" playlist skews heavily left in valance because it is described as "deep and dark. bold and bare," which is definitely not upbeat and happy. However, the playlists I created that I compare both of those to show fairly normal and not skewed variance. My playlist, Black, despite consisting of moody trap rap shows more normality in valence. This is also true with Blue compared to Golden Hour, although I would think my playlist would share a similar spike at low valence values. Even my Purple playlist, which I would consider having the saddest emotional undertones, ranks fairly normal in valence. This also holds true with the danceability score. Most of my playlists skew right in danceability, which makes sense because I love to dance! However, I wouldn't often put on a playlist like Black or Red to dance, because I typically dance to indie music instead or rap.

I believe these two trends very clearly shows a personal musical bias as compared to the Spotify algorithm. I tend to be a generally happy person, and thus it's clear that all my music skews towards higher valances, even when I seek to listen to sad music. Similarly, since I enjoy dancing, it appears my playlists skew towards high danceability values, even if I don't consider them the right mood for dancing. This is actually really fascinating to me, because I always thought that my music choices somewhat contrasted my typical mood. It also suggests that Spotify's algorithm tends to really focus on song's emotional characteristics for their playlists without that bias.

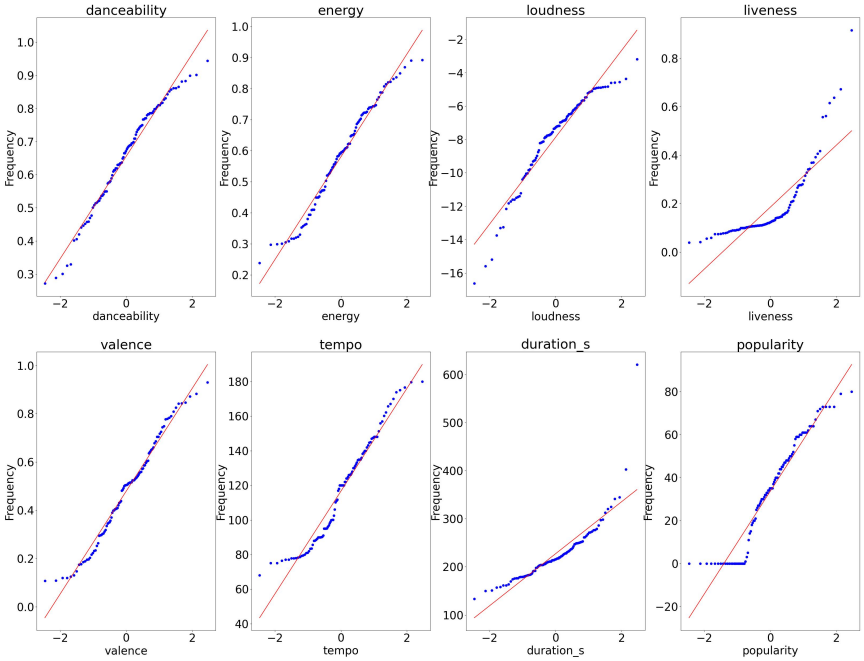
Some of the playlist comparisons appear to highlight that Spotify uses pure music characteristics better than I do. A good example is Red v. Beast Mode. For me the emotion characteristic of Red is loud and exciting rap music. Despite this, the tempo follows a somewhat uniform distribution. However, the Beast Mode playlist, which appears to be a playlist for getting hyped in the gym, shows a very large spike in tempo at 125 beats per minute. I don't play any musical instruments, and thus I often tend to rate music based more on personal preference and emotional characteristics than musical tendencies, and as a result I don't actively focus much on tempo. However, it's clear that Spotify's algorithm believes that 125 beats per minute is a popular tempo for exciting gym music, and seems to heavily prioritize that feature in their song choice for that playlist.



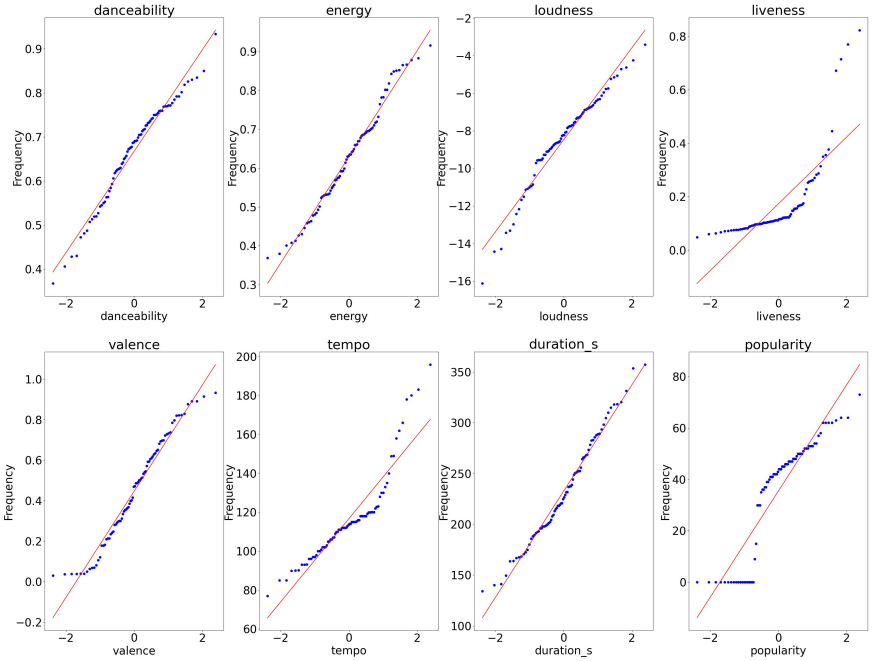
Feature Probability Plots for my life is a movie



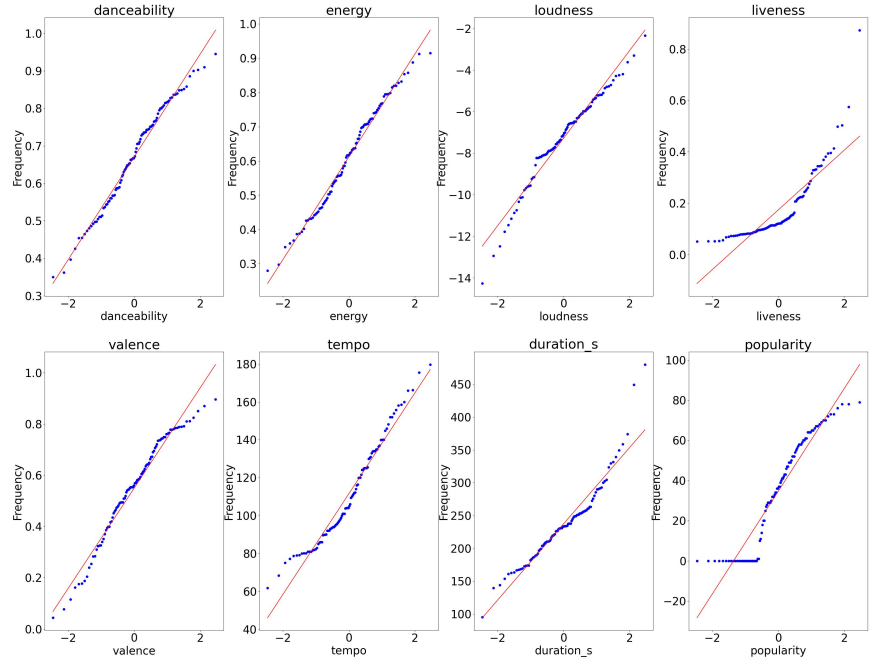
Feature Probability Plots for Blue



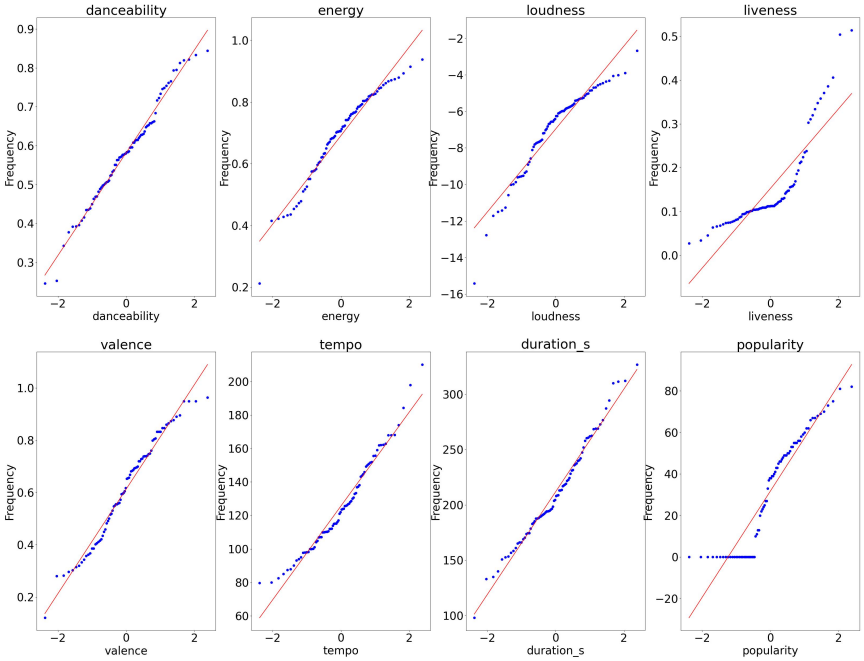
Feature Probability Plots for golden hour



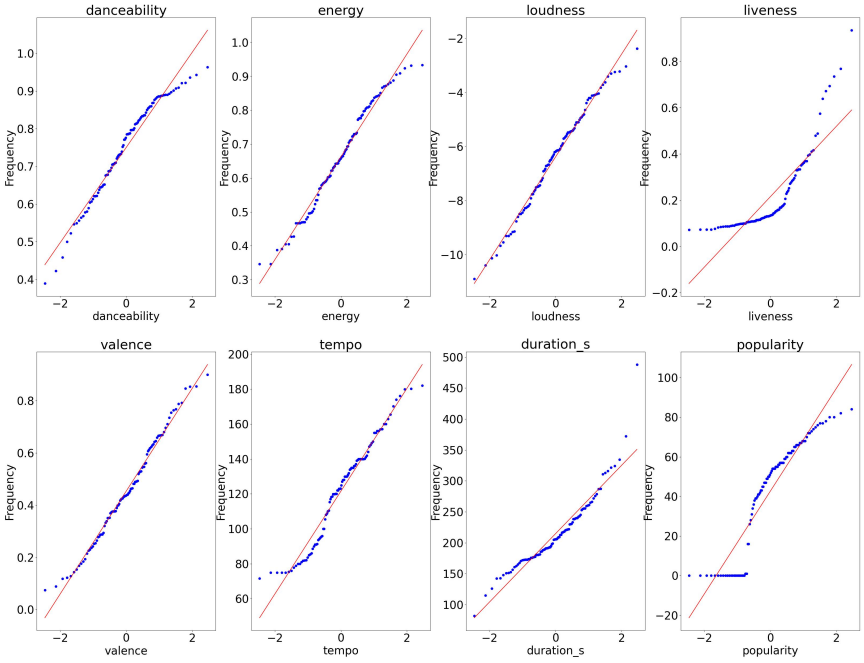
Feature Probability Plots for Green



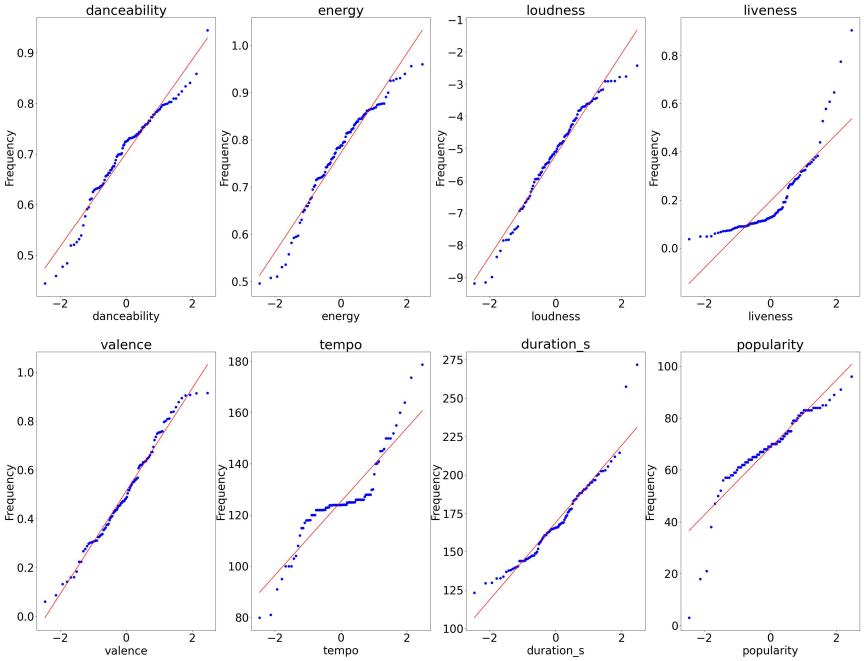
Feature Probability Plots for Surf Rock Sunshine



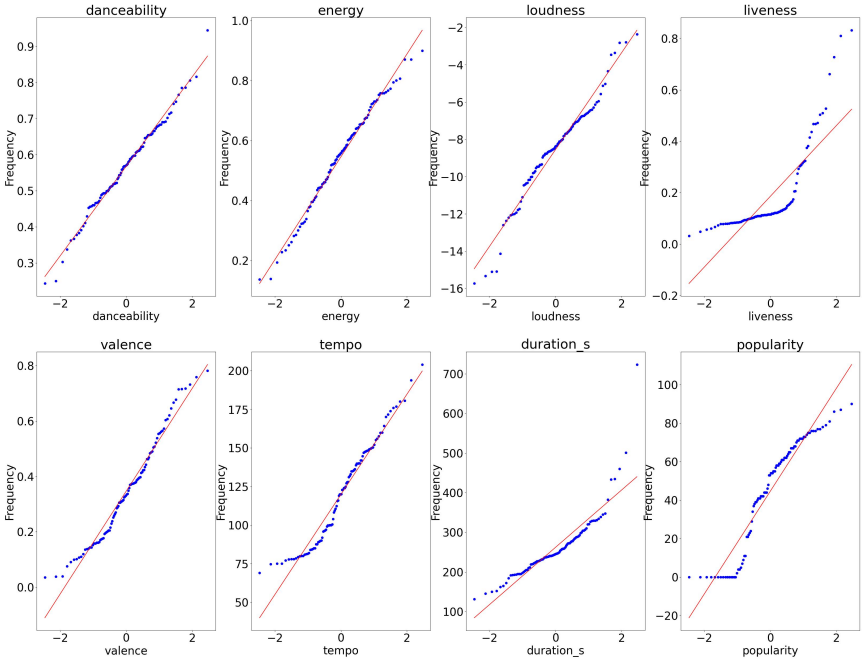
Feature Probability Plots for Red



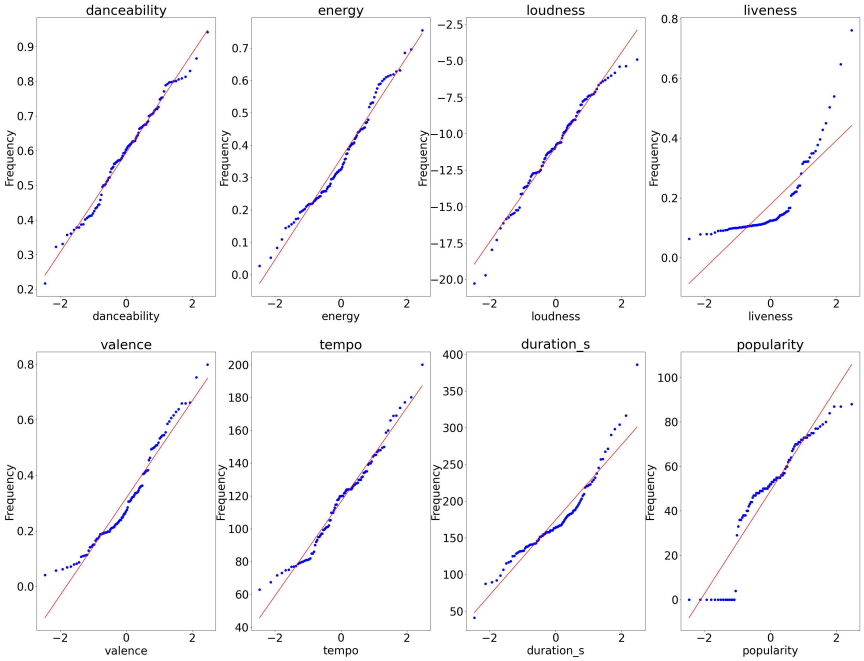
Feature Probability Plots for Beast Mode



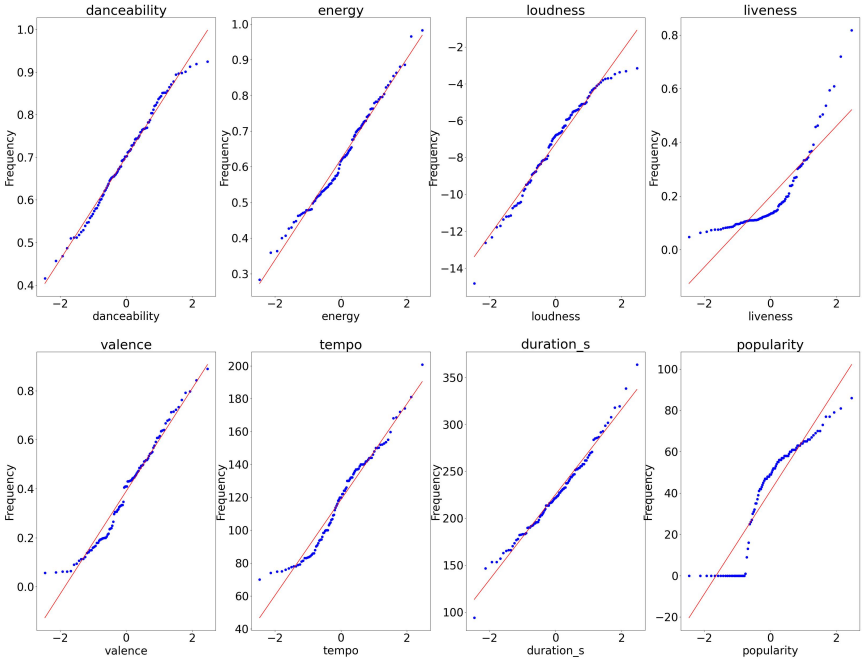
Feature Probability Plots for Purple

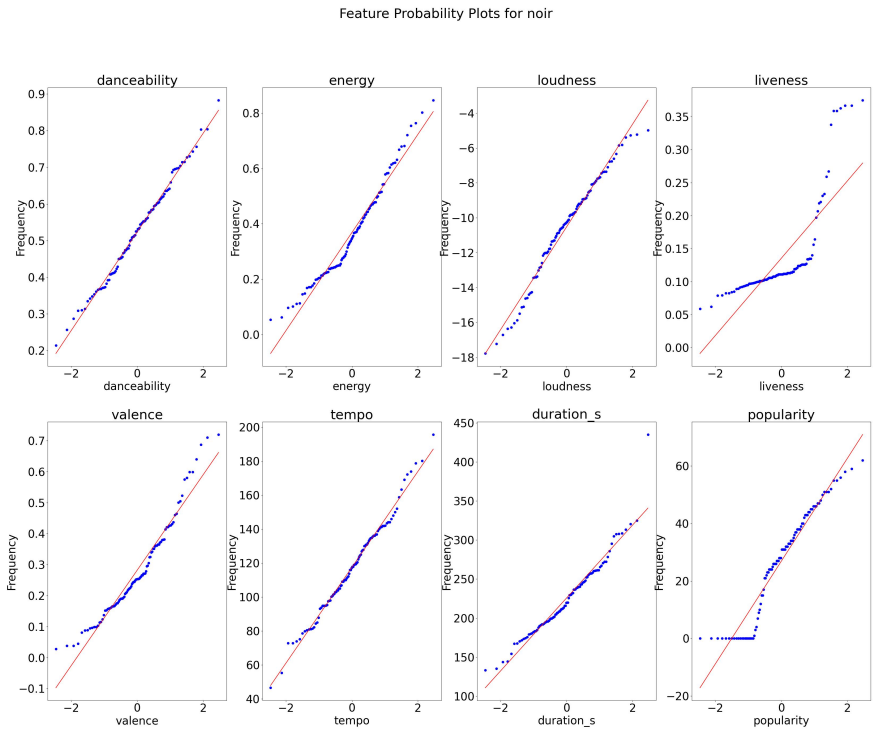


Feature Probability Plots for idk.



Feature Probability Plots for Black





The qualitative analysis gives us an array of interesting comparisons between the emotional characteristics of my playlists versus the Spotify generated ones. However, we should now turn to statistical tests to provide a quantitative analysis of how similar the playlists are. First, we take a look at the probability plots. As we recall from class, the closer the probability plot is to the straight line, the more normal the distribution is. Looking at the probability plots quickly, we can see some pretty distinct trends. Danceability, energy, valence, tempo, and the duration in seconds appear to be relatively normal across most of the playlists. It appears that loudness suffers from right skew, which makes me hesitate to suggest that it will be normal. Liveness and popularity are distinctly not normal. Liveness suffers from many low values with the occasional high outlier, and the popularity values suffer from many 0 values in the dataset, which skews both of these characteristics heavily to the left.

To test the normality, we will use the chi-squared test as discussed. The chi-square test will test the null hypothesis that the data comes from a normal distribution. Interestingly, I found a Python function withing the SciPy library that does a more advanced normal test based on D'Agostino and Pearson's test to combine skew and kurtosis to produce an omnibus test or normality [10]. Since this will be testing the same hypothesis, I decided to use that to provide a better result. Of note here is that we test for a 95% confidence value. This Python function returns the pvalue by comparing the normality statistic to the theoretical Gaussian distribution. We seek to turn this into a 0 or 1 value based on whether the test passes our 95% confidence value. We know that a p-value larger than 0.05 is thus not statistically significant, and indicates strong evidence for the null hypothesis. If the null hypothesis passes, we believe the data is normal with 95% confidence, and we report a value of 1, and if the test fails we report a value of 0. Note that each column represents one of the playlist pairs, labeled by their first playlist, for example: Yellow corresponds to the "Yellow" and "My Life is a Movie" playlist pair. The tuple of values similarly gives the normality test for each respective playlist in the pair.

Feature	Yellow	Blue	Green	Red	Purple	Black
Danceability	(1, 1)	(1, 1)	(1, 1)	(0, 0)	(1, 1)	(1, 1)
Energy	(0, 1)	(0, 1)	(1, 0)	(0, 0)	(1, 1)	(1, 1)
Loudness	(0, 0)	(0, 0)	(0, 0)	(1, 1)	(1, 1)	(1, 1)
Liveness	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)
Valence	(1, 0)	(0, 0)	(1, 0)	(1, 1)	(0, 0)	(1, 0)
Tempo	(1, 1)	(0, 0)	(0, 1)	(0, 0)	(1, 1)	(0, 1)
Duration	(0, 0)	(0, 1)	(0, 1)	(0, 0)	(1, 0)	(0, 0)
Popularity	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)	(0, 0)

Table 1 Normality Test Results

The results are somewhat surprising, and actually show that less of the features than I expected meet the 95% confidence level for normality. Danceability is normal across the board, except for the Red/Beast Mode playlists, which I am suprised has enough outliers to make this not normal. Energy appears to be normal for the Purple and Black playlists, not at all for red, and shows mixed results for Yellow, Blue, and Green. Loudness is not normal for Yellow, Blue, and Green, but is for Red, Purple, Black. This makes sense to me, the first three playlists are relatively relaxed and as a result are probably pretty skewed for loudness. As expected, both liveness and popularity are not normal at all. Valence shows not normal results for Purple and Blue, which is expected because both of these have a higher proportion of sad songs and are thus skewed. I am somewhat surprised that the Valence of my Green playlist is normal, since I would expect that to be skewed far to the right. Tempo shows a decent bit of non-normality, which initially surprised me. However, considering that for most of the playlists tempo shows a distinct peak means that even slight deviations can be considered outliers, thus ruining the normality which makes sense. Additionally, many of the duration results are not-normal, which is likely due to outliers caused by very long songs, since most of the songs are around the 2-3 minute range.

We that we have seen the results for normality in the features, we want to perform a t-test comparing the two playlists in each set of playlists to determine if they come from the same distribution with 95% confidence. One issue here though is that for a successful t-test, both the populations should be relatively Gaussian and have similar variances. Unfortunately, from the results of our normality tests, many of the playlists don't follow exactly Gaussian distributions. Although some of these come from a few outliers, the playlist sets that didn't have Gaussian distributions have (NN) next to them, standing for "Non-normal." We should consider these results with care.

Feature	Yellow	Blue	Green	Red	Purple	Black
Danceability	0	1	0	0 (NN)	0	1
Energy	0 (NN)	0 (NN)	1 (NN)	0 (NN)	0	0
Loudness	0 (NN)	0 (NN)	0 (NN)	0	0	0
Liveness	0 (NN)	0 (NN)	0 (NN)	0 (NN)	0 (NN)	0 (NN)
Valence	1 (NN)	0 (NN)	1 (NN)	0	0 (NN)	1 (NN)
Tempo	1	0 (NN)	0 (NN)	0 (NN)	1	0 (NN)
Duration	0 (NN)	0 (NN)	0 (NN)	0 (NN)	1 (NN)	0 (NN)
Popularity	0 (NN)	0 (NN)	0 (NN)	0 (NN)	0 (NN)	0 (NN)

Table 2 T-Test Results

Let's look first at the samples that actually pass the t-test, because those are the most interesting. First, the Blue and Golden Hour playlists pass for Danceability, which makes sense because I would expect both of these to not be very danceable playlists. The same holds true for the Black and Noir playlists. We also see similar tempo results for Yellow and My Life is a Movie, as well as Purple and Idk. I would expect the tempo results to be very similar across the playlist pairs because I think unconsciously the tempo actually plays a large part in the emotion characteristics, whether I actively listen for it or not. The result of the tests results for tempo were not Gaussian, which could have result in the

t-test failures. The only other passing tests are for Valence in the Green and Black sets, which is interesting. I would expect both of these playlist pairs to have pretty similar happiness values, because Green and Surf Rock Sunshine are both very happy, and Black and Noir are both pretty moody and sad. Unfortunately, since both of these features are quite skewed, they are not normal, and thus we cannot fully trust the results of the t-test. This concludes the analysis on emotional relate-ability between playlist pairs.

For the next analysis, we want to dive into the API statistics and try to create a linear model that predicts the song popularity. Since this analysis will be complex, I will only do it for two playlists, and discuss how the results might chance depending on the playlist in the conclusion. It's expected that the Spotify algorithm created playlists try to maximize popularity for their specific purpose because that equates to more streams, and of course more money. Thus, this analysis will be better performed on the Spotify playlists instead of my personal playlists. I believe that the "Noir" playlist will be the most interesting for this analysis. There are two reasons: first the API features will mostly normal for this playlist, except in categories where none of the playlists did well. Additionally, this playlist contains a lot of "underground" songs, which are songs that aren't necessarily popular. On the contrary, the "Beast Mode" playlist contains almost exclusively very popular songs, which makes sense for a gym playlist that may be played for a variety of people at once. Comparing how well the linear model behaves between these two playlists will give us good insight into how well the model performs depending on the initial popularity of the song.

First, we will create a correlation matrix between the API features. This will help to provide a preliminary estimate of how correlated two features are. For example, I expect that "energy" and "loudness" will be quite correlated. This doesn't necessarily mean that one of these predictors will be insignificant in the linear regression, but it might help explain any weird looking results in the final predictor coefficient values. The correlation matrix for each playlist is given below.

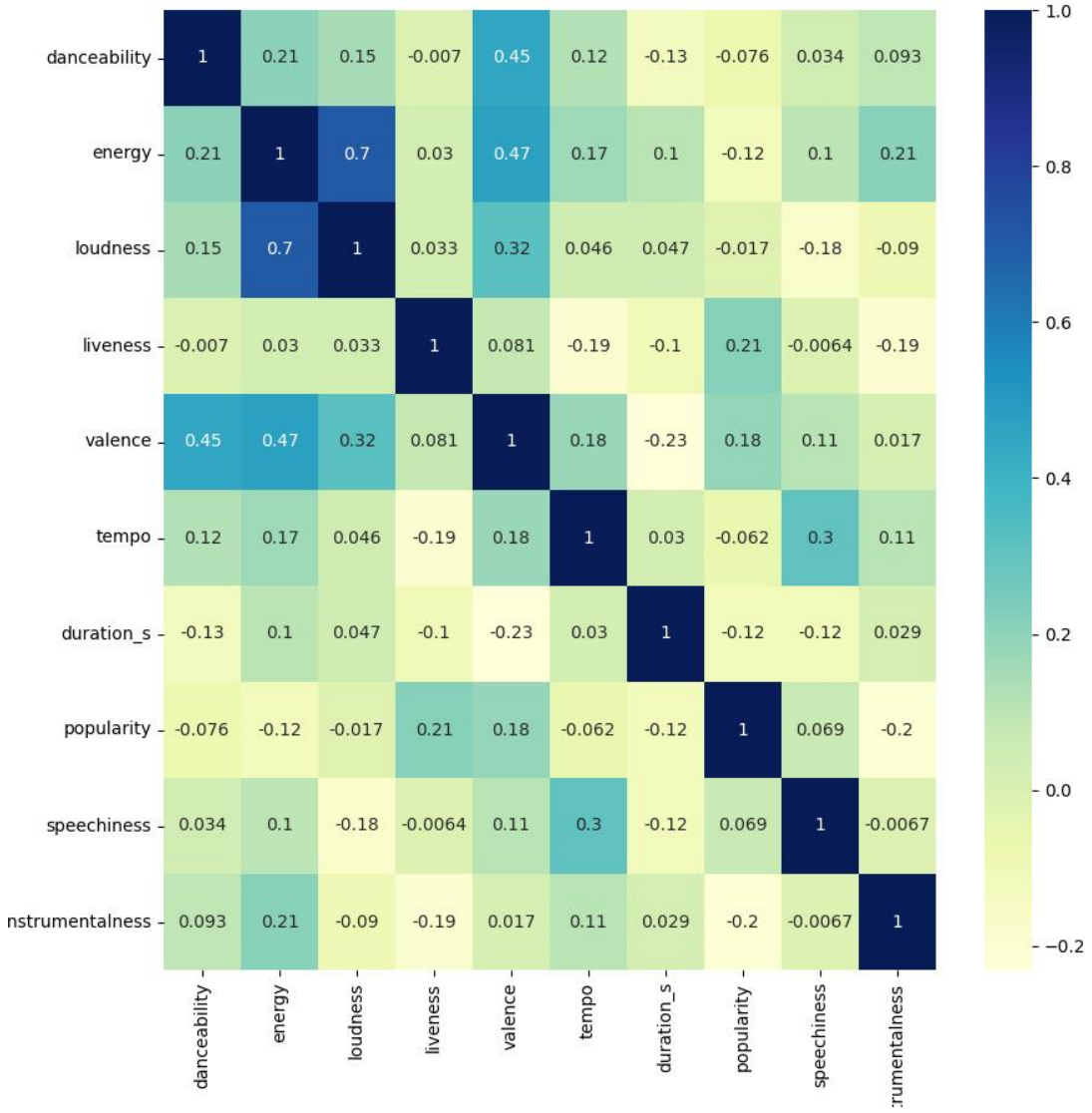


Fig. 7 Correlation Matrix for the Noir Playlist

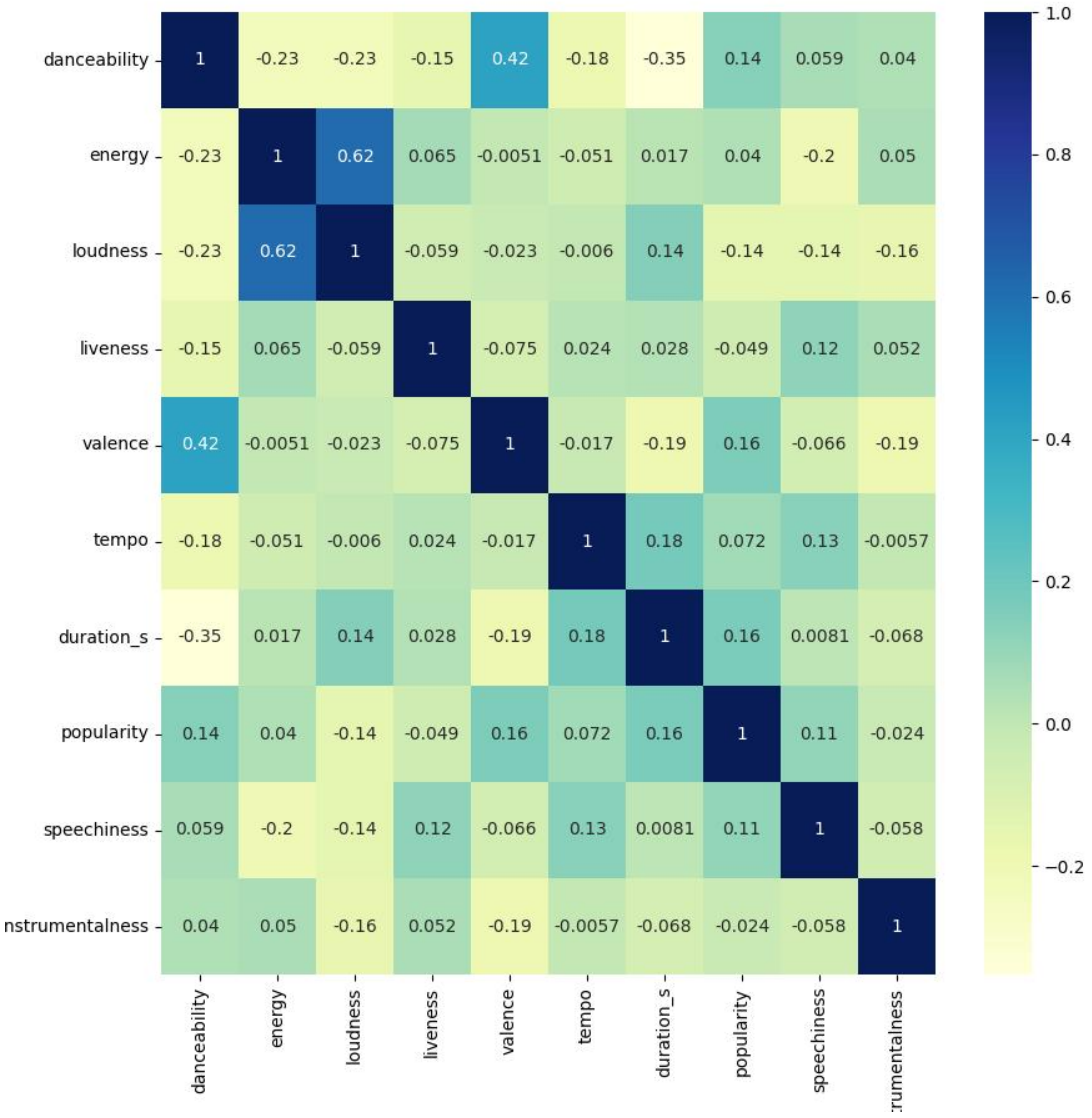


Fig. 8 Correlation Matrix for the Beast Mode Playlist

The correlation matrices give us interesting insight. Firstly, a fair bit of the API features are not correlated at all, which is good because it means these are all relatively unique characteristics of songs. However, there are some that show strong correlations. As expected, energy and loudness share high correlation for both the Noir and the Beast Mode playlists, so do valence and danceability. Who knew that happy songs were fun to dance to? Interestingly energy also corresponds highly with valence for the Noir playlist, but not at all for the Beast Mode playlist. What could cause this stark difference? My guess is that both energy and valence and low for noir, whereas energy is very high for Beast Mode but valence is fairly standard, thus they show stark differences within the playlist. The same is true for loudness and valence, although to a lesser degree. Essentially, playlists were features congregate near lower values tend to see correlations between API features that we might not necessarily assume are correlated. Since we included popularity on these correlation matrices, we can see a first guess estimate of how well the other parameters will be able to predict popularity. Unsurprisingly, popularity has very low correlation with most of the features, and those differ heavily between the Noir playlist and the Beast Mode playlist. I expect that the linear model will

not be a strong estimator of popularity due to the complexity of song characteristics and popularity, and this helps show that.

When working on the linear regression, I decided to do include the Purple playlist as well, because I wanted one with slightly more samples just to make sure low sample numbers weren't heavily worsening the results. Tables of the linear model coefficients, prediction v. actual popularity scatterplots, and residual histograms are given below.

Feature	**Coefficient Value**
Danceability	-21.211904463816023
Energy	-45.75373499427435
Loudness	1.2550910433209026
Liveness	46.82569425750833
Valence	48.77548860498746
Tempo	-0.016346340061510034
Duration	0.022161732449518468
Speechiness	47.1492382384632764
Instrumentalness	-1.6450915343288344

Table 3 Noir Linear Model Coefficients

Feature	**Coefficient Value**
Danceability	-1.55254753699216951
Energy	29.562273535543895
Loudness	-2.89340950322169
Liveness	-3.615856113392882
Valence	17.789693349303445
Tempo	0.18560960957094697
Duration	0.07121906468145056
Speechiness	51.90583672827054
Instrumentalness	-2.1659470861914514

Table 4 Beast Mode Linear Model Coefficients

Feature	**Coefficient Value**
Danceability	25.142681307435414
Energy	2.1166228416843116
Loudness	1.8119583622359945
Liveness	-15.895057678831066
Valence	-10.137829470414914
Tempo	-0.035125688325328586
Duration	0.005709786694728658
Speechiness	11.677301406280296
Instrumentalness	-4.491597229212287

Table 5 Purple Linear Model Coefficients

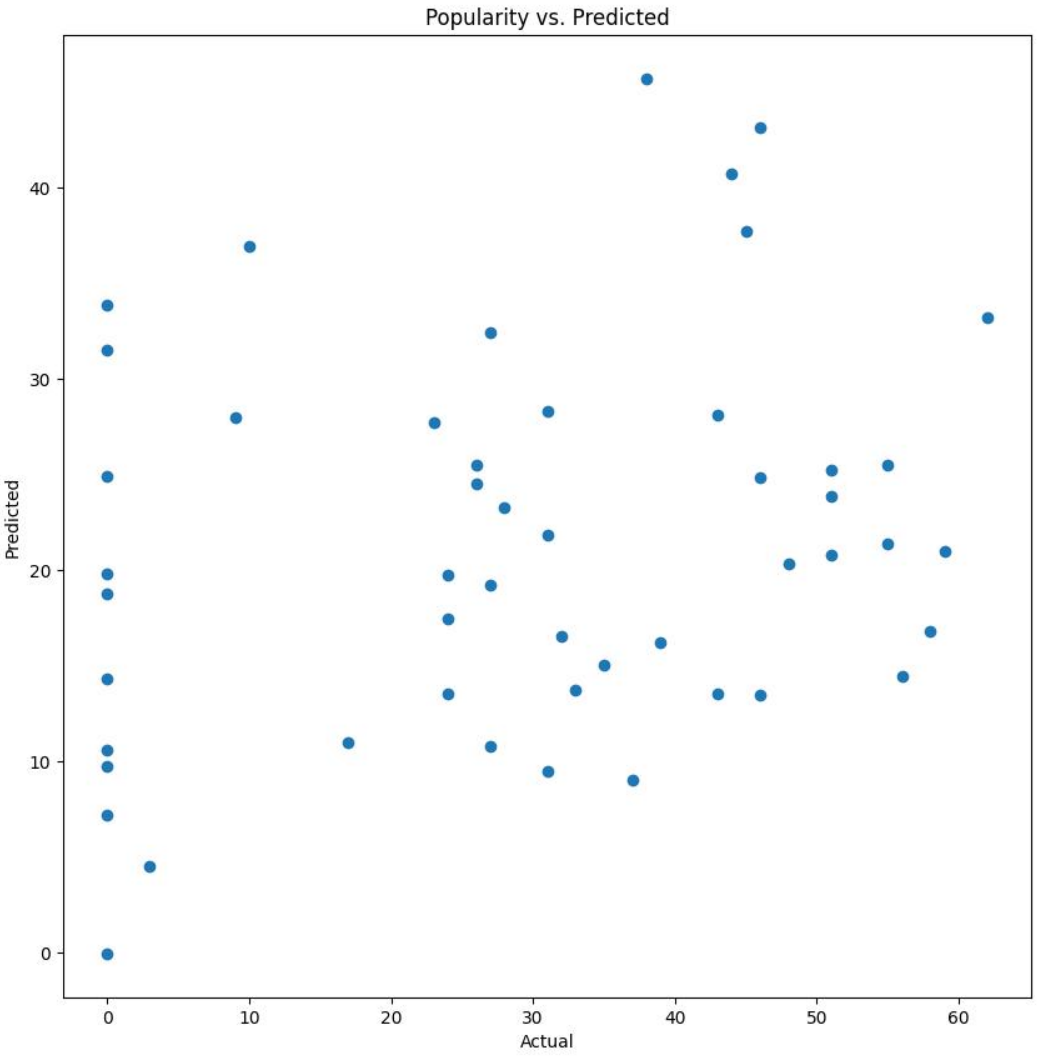


Fig. 9 Noir Playlist Popularity Predictions v. Actual

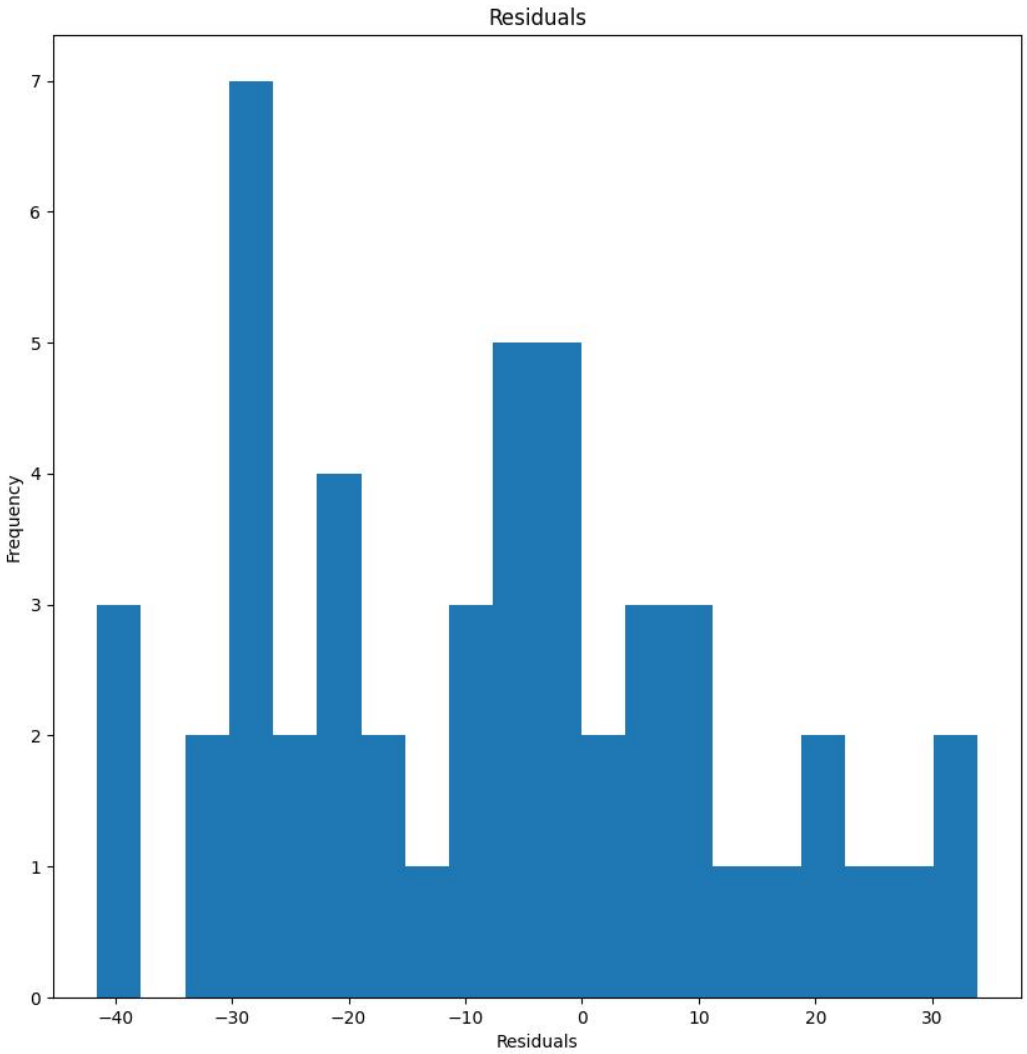


Fig. 10 Noir Playlist Residual Histogram

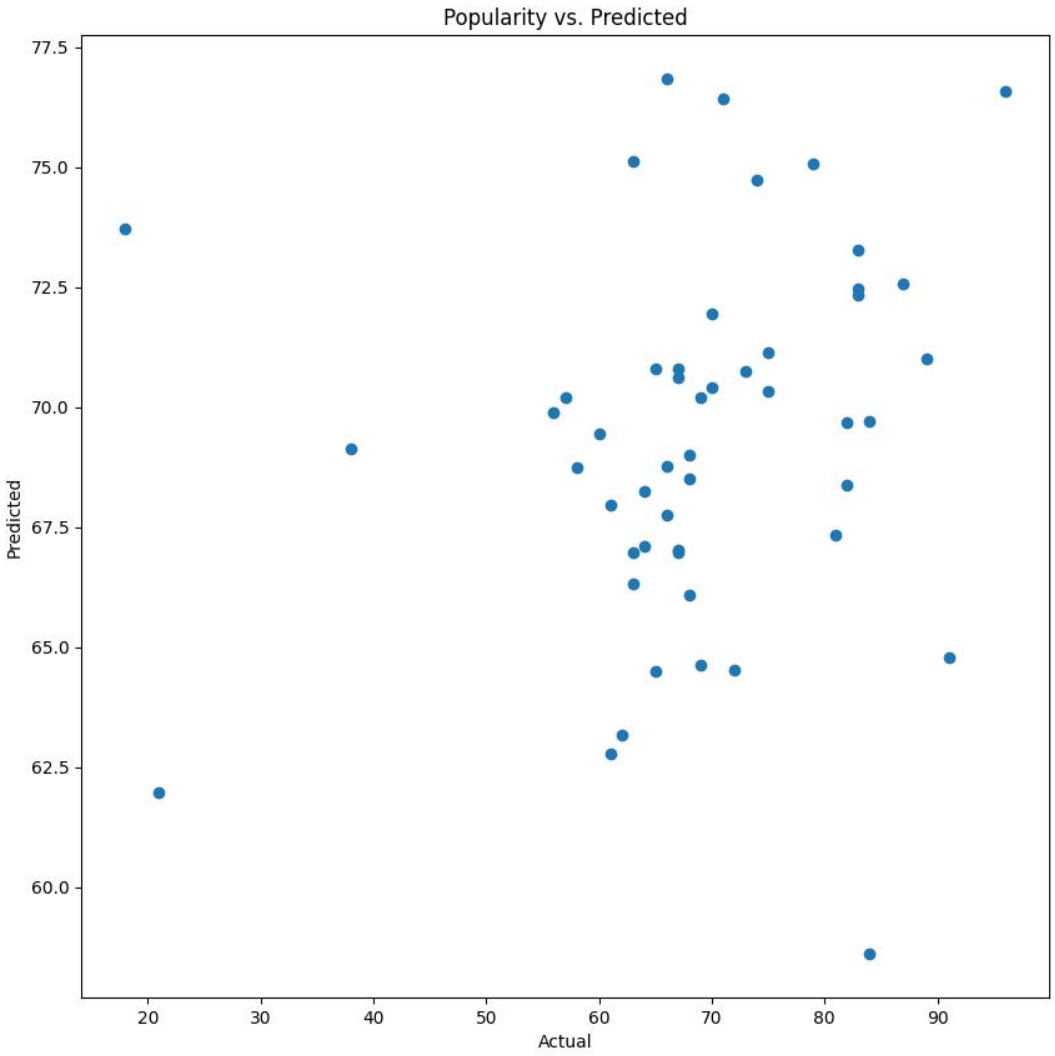


Fig. 11 Beast Mode Playlist Popularity Predictions v. Actual

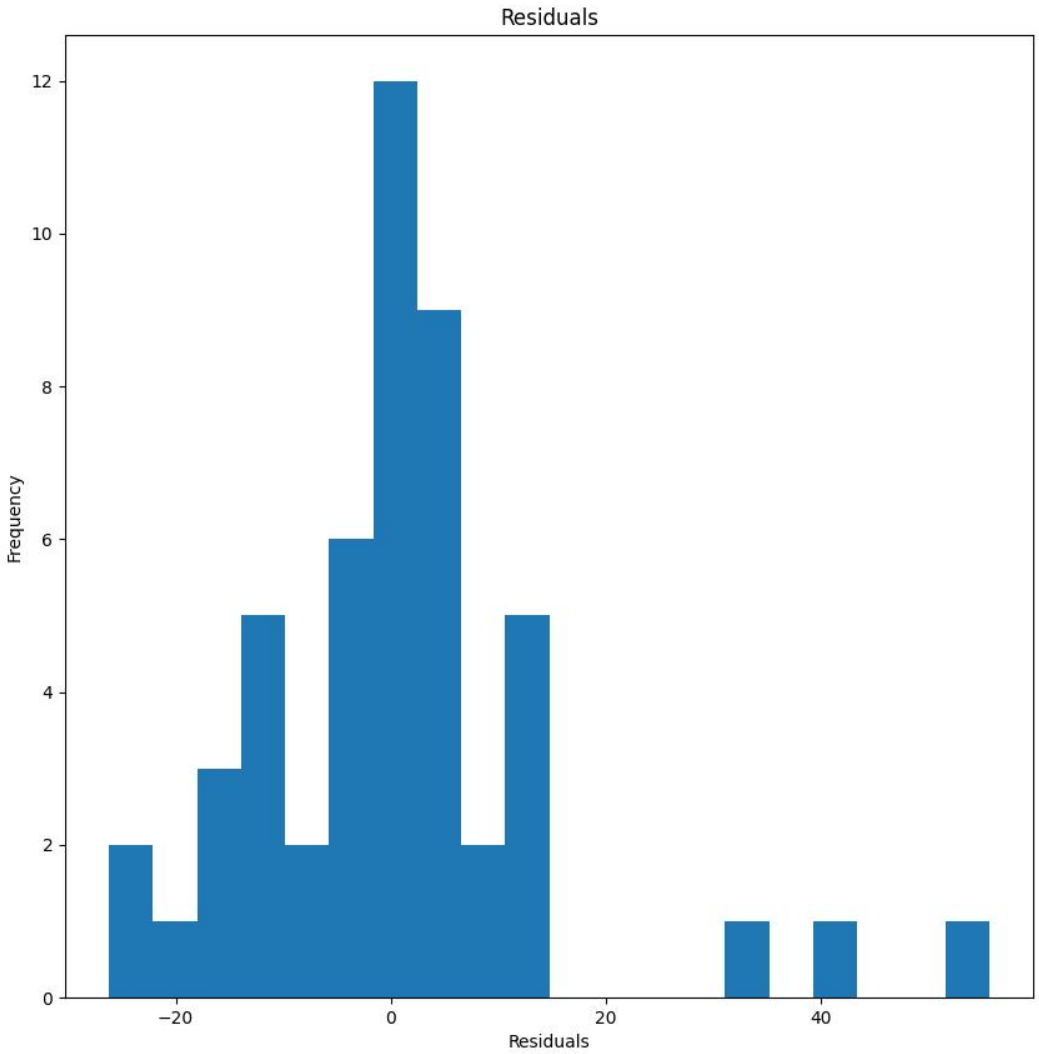


Fig. 12 Beast Mode Playlist Residual Histogram

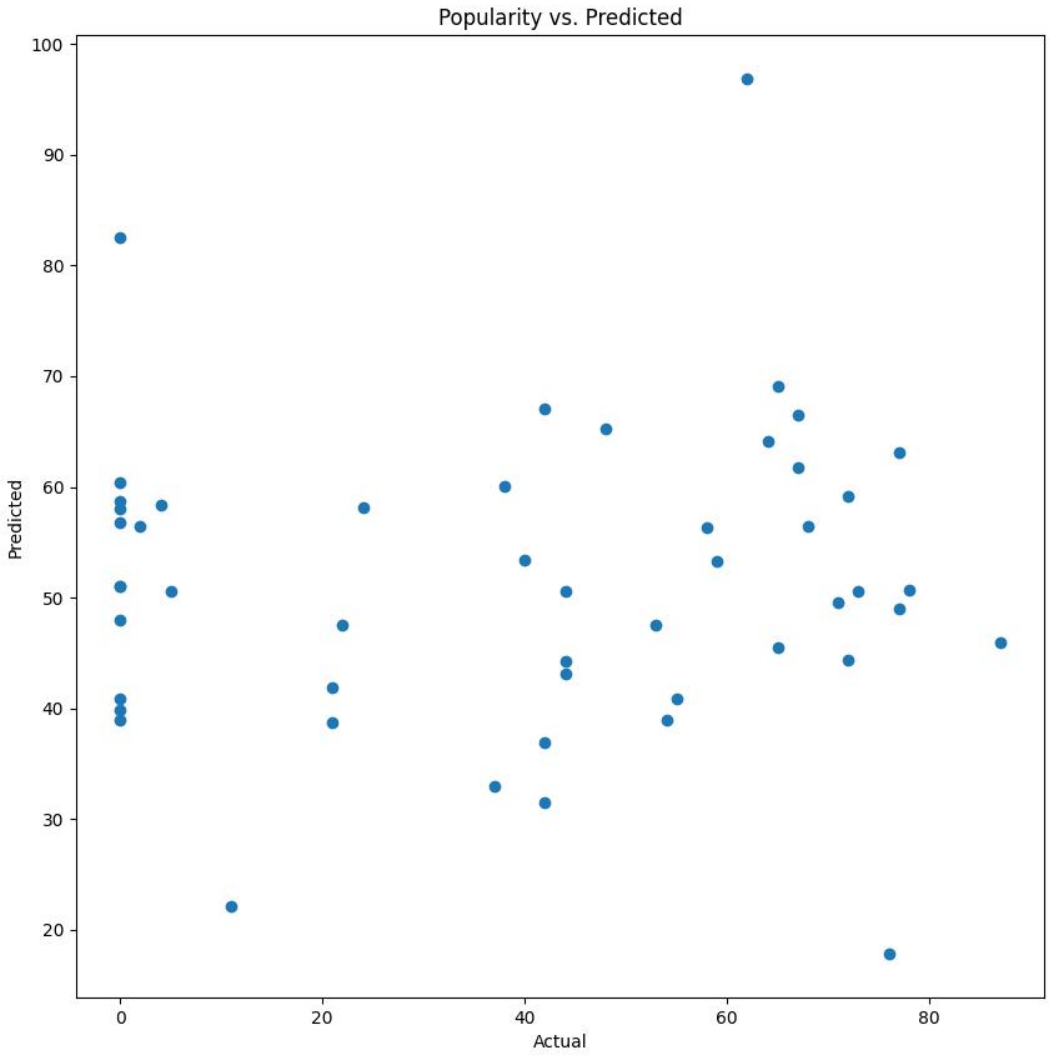


Fig. 13 Purple Playlist Popularity Predictions v. Actual

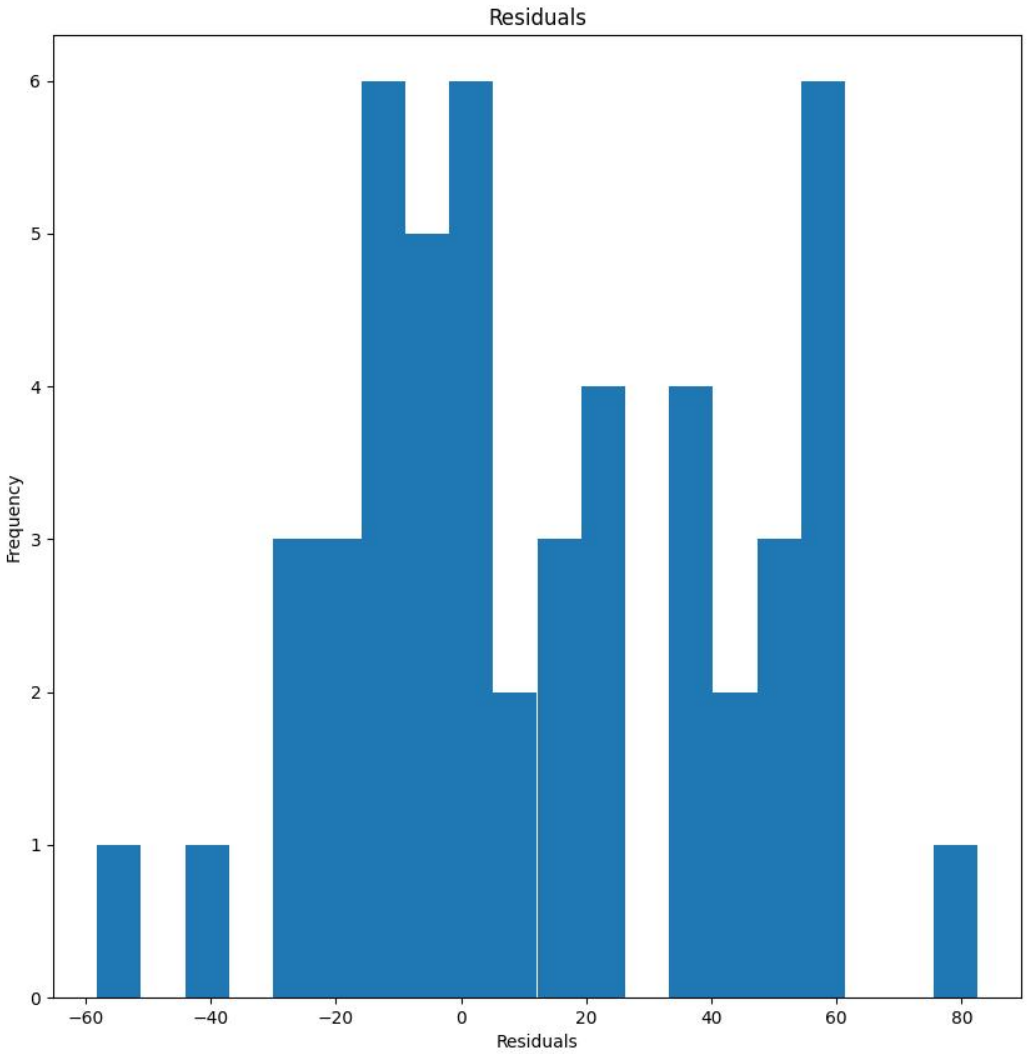


Fig. 14 Purple Playlist Residual Histogram

Across the board the linear model poorly estimates song popularity values for each playlist. Interestingly, the linear models are completely different for the different playlists. The linear model for Noir predicts low popularity despite the actual popularity ranging from 0 to 50, with an R-squared value of -0.16. The Beast Mode playlist similarly predicts poorly, a fair bit of the actual popularity values are between 60 to 80 and so are the predicted values, but the outliers lead to a R-squared value is 0.02. For my personal playlist, the predictor is terrible, predicting a popularity score of around 35-55 despite the actual popularity ranging anywhere from 0 to 90. The R-squared value for Purple is -0.35. The residuals histograms help to show how bad the predictions are. For all three playlists there are significant residuals away from 0, and none of these histograms are even close to normality. It would be tempting to remove outliers here, but these are not outliers caused to poor data collection, these are genuine predictions that are far off. If we were creating a model to predict popularity and we wanted to actually send it to an artist or recording studio, telling them what led to popularity, our models would be very wrong. The coefficients provide the most quantitative analysis of this failure in our linear model. The feature coefficients change drastically depending on the nature of the music, which actually

makes a lot of sense. For Noir, danceability and energy have negative coefficient values because this playlist consists of sad music. However, having a low valence value is probably heavily important here and thus the coefficient value for that is pretty high. On the contrary, Beast Mode values energy and valence as well, but doesn't care much about things like liveness or danceability. Some of the feature coefficients do strike me as strange though. I am surprised that liveness has such a high value for Noir, but perhaps this more underground playlist values live songs. I am also surprised at how little duration scores across all three playlists, since it seems like shorter songs have been becoming more normal in the industry, although this may be because of the value of individual streams as opposed to pure song popularity. Another thing I noticed is that it looks like Speechiness is valued highly in Spotify algorithm generated playlists, but not in my personal playlists. I tend to listen to a lot of music, both with and without lyrics, so this makes sense. Overall here, our linear model fails to helpfully predict popularity, as expected.

V. Summary & Conclusions

Overall I found this project to be engaging and fun, while cementing some of the fundamental skills that I've learned through this course. For the emotional analysis, I was surprised at the results. In particular, I was surprised at both the bias in my decisions as well as the normality of the feature distributions. In general it appears that my music choices, regardless of what emotional category I slot them into, follow relatively similar trends. Essentially, it appears that I enjoy a certain kind of music: upbeat, happy, high energy, easy to dance too, and relatively popular. Even when I attempt to separate songs into different playlists, the emotional differences are small from my baseline song preferences. However, Spotify's algorithm seems to heavily stress certain features depending on what they want a playlist to achieve. Beast mode consists of highly popular songs at a specific tempo, which would fit perfectly for a gym setting. Sadder playlists like "Idk" and Noir are much more left skewed in their valence and energy, showing that when they want to create a sad playlist, they make sure these features are stressed compared to other less effective features like popularity as it relates to being sad. My guess is that this is representative of the tuning parameters used to create these playlists. Each feature is individually tuned for the playlist's goals, whereas my playlists are created from stumbling upon songs I already like and then slotting them in based on slight emotional divergence.

The results of the normality test and the t-test were disappointing, but in some ways expected. The truth is that many of these API features are skewed and suffer from outliers. Simply put, songs are so diverse that it can be expected they don't follow perfect distributions such as other singular datasets. The t-tests did return that some of my playlists matched well with the Spotify algorithm generated playlists in specific characteristics. However, across the board these results show that there are stark differences in the way I create my playlists compared to how the Spotify algorithm creates them, which is fair. There's a reason why I find benefits from both my personal playlists as well as the algorithm ones. I think the main difference is that the algorithm created playlists focus heavily on a specific feature, such as the nearly uniform tempo results for "Beast Mode." Whereas my playlists show slight deviations in emotional characteristics from my base listening habits.

Jumping into the correlation showed some interesting insight into these API features. We saw that a basic multiple linear regression model to predict popularity is a failure. This makes complete sense. Music is intensely complicated, and nobody can really crack why people like certain music and why people don't. This is why Spotify machine learning engineers make 6 figures and get hired in droves, because even getting somebody to stumble upon one song they absolutely love is a difficult feat.

One thing to note here is that we had to predict "popularity", which is a Spotify generated score from 0 to 100. The issue here is that what goes into this popularity score is vague, and the Spotify API doesn't provide a more direct representation of song popularity such as number of streams. Unfortunately their popularity score also includes a temporal aspect, which I am not analyzing in my linear regression because Spotify does not provide information on how the time since the streams affects this popularity score. Since I listen to a wide variety of songs, many not very popular, I think this score may be affecting the linear regression. Perhaps if we could predict number of streams, we may see slightly better results.

Additionally, there are a wide range of other factors that affect popularity that our linear model does not account for. For example, we make no attempt to include the genre of songs here. However from our initial analysis it seems that genre is quite important, since the linear model coefficients change so heavily between the Noir and Beast Mode

playlists. We also do not include any geographic information, which certainly influence the popularity of different types of music. Additionally, popularity of songs also changes over time as certain trends catch on and older trends start to become overplayed and boring. The conclusion here is that the techniques we learned in class may be useful for many types of scientific data, but start to break down as we explore much more complicated data, especially when the sample size is small and the data often fails to be Gaussian. Despite this, the project was an absolute blast and I am thrilled with how interesting the results were. Now I want to take a machine learning class so I can improve the prediction algorithm, although I doubt I'll get anywhere near Spotify's skills without joining the company!

References

[1] https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

[2] https://developer.spotify.com/documentation/web-api/

[3] https://www.analyticsinsight.net/tech-for-enjoying-music-heres-how-spotify-uses-big-data/: :text=Spotify%20uses%20big%20data%20not,users%20a

[4] https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3557124

[5] https://cdr.lib.unc.edu/concern/masters$_papers/ns064961b$

[6] https://www.researchgate.net/publication/334987891$_Temporal_Trends_in_Music_Popularity_-_AQuantitative_analysis_of_Spotify_API_data$

[7] https://arxiv.org/ftp/arxiv/papers/2108/2108.02370.pdf

[8] https://www.semanticscholar.org/paper/International-Music-%3A-An-Analysis-of-the-Suh/9b8c886384036f21fa2d8c84970f920d8786588f

[9] https://www.semanticscholar.org/paper/A-Longitudinal-Model-for-Song-Popularity-Prediction-%C3%87imen-Kayis/7e5d0f280b4aa46c23cf27488b5a56ea32b19430extracted

[10] D'Agostino, R. and Pearson, E. S. (1973), "Tests for departure from normality", Biometrika, 60, 613-622

VI. Appendix A: Code

A. Acquiring Data

```
# importing the necessary packages
import pandas as pd
import spotipy
import spotipy.util as util

from spotipy.oauth2 import SpotifyClientCredentials

def getSpotifyToken():
    # credentials for spotify
    sp = spotipy.Spotify()

    # setting up authorization
    cid ="f395b1d0ba9b4f9f9d83e833bf1267ff"
    secret = "e68f3c64c84648a594e00e4b95801420"
    redirect_uri ="http://localhost:7777/callback"
    username = "jashanxchopra@gmail.com"

    # scope to get the user library
    scope = 'user-library-read'

    # get the spotify token
    token = util.prompt_for_user_token(username, scope, cid, secret, redirect_uri)

    # also grab Spotify client creds
    client_credentials_manager = SpotifyClientCredentials(client_id=cid, client_secret=secret)
    sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

```
    return token, sp

def call_playlist(creator, playlist_id, sp):
    # gets information from a specific playlist
    # sourced from: https://www.linkedin.com/pulse/extracting-your-fav-playlist-info-spotifys-api-samantha-jones

    # setup dataframe
    features = ["artist","album","track_name","track_id",
                "danceability","energy","key","loudness",
                "mode", "speechiness","instrumentalness",
                "liveness","valence","tempo", "duration_ms",
                "time_signature", "acousticness"]

    playlist_df = pd.DataFrame(columns = features)

    # get the track details
    playlist = sp.user_playlist_tracks(creator, playlist_id)
    tracks = playlist['tracks']['items']
    for track in tracks:
        playlist_features = {}

        # Get metadata
        playlist_features["artist"] = track["track"]["album"]["artists"][0]["name"]
        playlist_features["album"] = track["track"]["album"]["name"]
        playlist_features["track_name"] = track["track"]["name"]
        playlist_features["track_id"] = track["track"]["id"]

        # add the track's popularity
        playlist_features["popularity"] = track["track"]["popularity"]

        # Get audio features
        audio_features = sp.audio_features(playlist_features["track_id"])[0]
        for feature in features[4:]:
            playlist_features[feature] = audio_features[feature]

        # add the playlist name
        playlist_features["playlist_name"] = playlist["name"]

        # convert duration_ms to seconds for plotting purposes later
        playlist_features["duration_s"] = playlist_features["duration_ms"]/1000

        # Concat the dfs
        track_df = pd.DataFrame(playlist_features, index = [0])
        playlist_df = pd.concat([playlist_df, track_df], ignore_index = True)

    return playlist_df

if __name__ == '__main__':
    # get spotify token
    token, sp = getSpotifyToken()

    # use tokedef call_playlist(creator, playlist_id, sp):n to aquire playlist information
    yellow = call_playlist("JashanChopra", "1Y9jEeQLlt46VFEeYzZ3zn?si=ef4e02410b1849cd", sp)
    blue = call_playlist("JashanChopra", "5g5x0aWpm2Uh5Nc0JbpIhf?si=47a4ec21ea2d4505", sp)
    green = call_playlist("JashanChopra", "3sx2rV67D7Ro4e6RgO0Amd?si=89ebf3803b184115", sp)
    red = call_playlist("JashanChopra", "1sA0ICkMKvRf5ghOPJ4Buq?si=bca99e339c294581", sp)
    black = call_playlist("JashanChopra", "0ilppVom6odKC1YhtlaOX4?si=ff531d8a78034799", sp)
    purple = call_playlist("JashanChopra", "4apY0heTn8caTtJgyYE188?si=e5d341fccebd4cfd", sp)

    # get spotify playlists of similar ideas
    movies = call_playlist("Spotify", "37i9dQZF1DX4OzrY981I1W?si=2c3b19568cdb4761", sp)         # compare to yell
    goldenhour = call_playlist("Spotify", "37i9dQZF1DWUE76cNNotSg?si=76c7b19eb0cf45a9", sp)     # compare to blue
    surfrock = call_playlist("Spotify", "37i9dQZF1DWYzpSJHStHHx?si=40e993be4845415e", sp)       # compare to gree
    workout = call_playlist("Spotify", "37i9dQZF1DX76Wlfdnj7AP?si=a473200f0c4a46d2", sp)        # compare to red
    dark = call_playlist("Spotify", "37i9dQZF1DX9LT7r8qPxfa?si=8ee35e5d7a3549c6", sp)           # compare to blac
    idk = call_playlist("Spotify", "37i9dQZF1DX59NCqCqJtoH?si=487debe7b74048c1", sp)            # compare to purp

    # export to .csv for later ease of use
```

```
    data = [yellow, blue, green, red, black, purple, movies, goldenhour, surfrock, workout, dark, idk]
    names = ["yellow", "blue", "green", "red", "black", "purple", "movies", "goldenhour", "surfrock", "workout",
    for idx, playlist  in enumerate(data):
        playlist.to_csv("./data/playlist_" + names[idx] + ".csv")
```

B. Loading Data

```
import pandas as pd
import os

def load_data():
    # Load the data for each of our csv files
    # This is just to save time and not have to call the Spotify API every time

    # For each .csv file in the directory, load the .csv file back into a pandas dataframe
    data = {}
    filenames = []
    for file in os.listdir("./data"):
        if file.endswith(".csv"):
            # get the name and read the csv
            filenames.append(file)
            df = pd.read_csv("./data/" + file)

            # Add the dataframe to the dictionary
            data[file] = df

    return data, filenames
```

C. Analysis

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import scipy.stats as stats
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

from loadData import load_data


# The first analysis is on the emotional characteristics
def AnalysisOne():
    features = ["danceability","energy","loudness","liveness","valence","tempo","duration_s","popularity"]

    # load playlist data
    data = load_data()

    # Array of playlist names for indexing and actual playlists
    playlists = data[0]
    names = data[1]

    # Create a hash of playlist tuples
    pairs = [(playlists[names[0]], playlists[names[6]]),
            (playlists[names[1]], playlists[names[7]]),
            (playlists[names[2]], playlists[names[8]]),
            (playlists[names[3]], playlists[names[9]]),
            (playlists[names[4]], playlists[names[10]]),
            (playlists[names[5]], playlists[names[11]])]
    pair_names = names[0:6]

    # dataframes to hold chi-squared and t-test results
    chi_results = pd.DataFrame(index=np.arange(len(features)), columns=pair_names)
```

```
t_results = pd.DataFrame(index=np.arange(len(features)), columns=pair_names)
stat_results = pd.DataFrame(index=np.arange(len(features)), columns=pair_names)

# adjust font size for all plt plots
matplotlib.rcParams.update({'font.size': 30})

# loop over each pair in pairs
for index, pair in enumerate(pairs):
    # get the two playlists
    playlist1 = pair[0]
    playlist2 = pair[1]

    # loop over the features and plot histograms for each feature
    fig, axs = plt.subplots(2, 4, figsize=(40, 30))
    for feature in features:
        plt.subplot(2, 4, features.index(feature) + 1)
        plt.hist(playlist1[feature], bins=20, alpha=0.5, label=playlist1["playlist_name"][0])
        plt.hist(playlist2[feature], bins=20, alpha=0.5, label=playlist2["playlist_name"][0])
        plt.legend()
        plt.title(feature)
        plt.xlabel(feature)
        plt.ylabel("Frequency")

        # place mean, mode, and standard deviation for both playlists in a dataframe
        mean = (np.mean(playlist1[feature]), np.mean(playlist2[feature]))
        mode = (stats.mode(playlist1[feature]), stats.mode(playlist2[feature]))
        std = (np.std(playlist1[feature]), np.std(playlist2[feature]))

        # place mean, mode, std into a single array and then into the dataframe
        stats_array = np.array([mean, mode, std], dtype=object)
        stat_results.iloc[features.index(feature), index] = stats_array

        # perform a chi-squared test on each histogram
        chi_squared, p_value = stats.normaltest(playlist1[feature])
        chi_squared_2, p_value_2 = stats.normaltest(playlist2[feature])

        # turn the p value(s) into a 1 if it is greater than 0.05
        p_value = 1 if p_value > 0.05 else 0
        p_value_2 = 1 if p_value_2 > 0.05 else 0
        chi_results.iloc[features.index(feature), index] = (p_value, p_value_2)

        # perform a t-test on each histogram if the chi-square result is normal
        if p_value == 1 and p_value_2 == 1:
            t_statistic, p_value = stats.ttest_ind(playlist1[feature], playlist2[feature])
            p_value = 1 if p_value > 0.05 else 0
            t_results.iloc[features.index(feature), index] = p_value
        else:
            t_results.iloc[features.index(feature), index] = str(p_value) + "(NN)"

    # save each plot to a jpeg and put them in a /images/ folder
    fig.suptitle("Feature Histograms for " + playlist1["playlist_name"][0] + " and " + playlist2["playlist_na
    plt.savefig('./images/' + playlist1["playlist_name"][0] + '_histogram.jpeg')

    # loop over the features and plot a probability plot for playlist 1
    fig, axs = plt.subplots(2, 4, figsize=(40, 30))
    for feature in features:
        # plot a histogram of each feature in the subplot
        plt.subplot(2, 4, features.index(feature) + 1)
        stats.probplot(playlist1[feature], dist="norm", plot=plt)
        plt.title(feature)
        plt.xlabel(feature)
        plt.ylabel("Frequency")
    # save each plot to a jpeg and put them in a /images/ folder
    fig.suptitle("Feature Probability Plots for " + playlist1["playlist_name"][0])
    plt.savefig('./images/' + playlist1["playlist_name"][0] + '_probplot.jpeg')

    # loop over the features and plot a probability plot for playlist 2
    fig, axs = plt.subplots(2, 4, figsize=(40, 30))
```

```
        for feature in features:
            # plot a histogram of each feature in the subplot
            plt.subplot(2, 4, features.index(feature) + 1)
            stats.probplot(playlist2[feature], dist="norm", plot=plt)
            plt.title(feature)
            plt.xlabel(feature)
            plt.ylabel("Frequency")
        # save each plot to a jpeg and put them in a /images/ folder
        fig.suptitle("Feature Probability Plots for " + playlist2["playlist_name"][0])
        plt.savefig('./images/' + playlist2["playlist_name"][0] + '_probplot.jpeg')

    # save the chi-squared and t-test results, and statistics to a csv
    chi_results.to_csv('./results/chi_results.csv')
    t_results.to_csv('./results/t_results.csv')
    stat_results.to_csv('./results/stat_results.csv')

# The second analysis will look at each API feature in more detail and produce a linear regression
def AnalysisTwo():
    features = ["danceability","energy","loudness","liveness","valence","tempo",
    "duration_s","popularity","speechiness","instrumentalness"]

    features2 = ["danceability","energy","loudness","liveness","valence","tempo",
    "duration_s","speechiness","instrumentalness"]

    # load playlist data
    data = load_data()

    # Array of playlist names for indexing and actual playlists
    playlists = data[0]
    names = data[1]

    # the two playlists I care about for this analysis
    noir = playlists[names[9]]
    beast = playlists[names[10]]
    # add purple to see with more samples...
    purple = playlists[names[5]]

    # loop over those two playlists
    for playlist in [noir, beast, purple]:

        # first we need to create a dataframe of the features and their values for the playlist
        data = {}
        for feature in features:
            data[feature] = playlist[feature]
        df = pd.DataFrame(data, columns=features)

        # create correlation matrix plots
        corrMatrix = df.corr()
        plt.figure(figsize=(10, 10))
        sns.heatmap(corrMatrix, annot=True, cmap="YlGnBu")

        # save the heatmap
        plt.savefig('./images/' + playlist["playlist_name"][0] + '_correlation_matrix.jpeg')

        # create a multiple linear regression for popularity from the other features
        X = df.drop(columns=["popularity"])
        y = df["popularity"]
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
        reg = LinearRegression().fit(X_train, y_train)
        y_pred = reg.predict(X_test)

        # create a scatter plot of the actual vs predicted values
        plt.figure(figsize=(10, 10))
        plt.scatter(y_test, y_pred)
        plt.xlabel("Actual")
        plt.ylabel("Predicted")
        plt.title("Popularity vs. Predicted")
        plt.savefig('./images/' + playlist["playlist_name"][0] + '_popularity_regression.jpeg')
```

```
        # print the r squared value for the linear model
        print("R squared value for " + playlist["playlist_name"][0] + ": " + str(reg.score(X_test, y_test)))

        # plot a histogram of the residuals
        plt.figure(figsize=(10, 10))
        plt.hist(y_pred - y_test, bins=20)
        plt.xlabel("Residuals")
        plt.ylabel("Frequency")
        plt.title("Residuals")
        plt.savefig('./images/' + playlist["playlist_name"][0] + '_residuals.jpeg')

        # save the linear regression coefficients to a csv file
        df = pd.DataFrame(reg.coef_, columns=["coefficients"])
        df["features"] = features2
        # switch the order of the features and coefficients columns
        df = df[["features", "coefficients"]]
        df.to_csv('./results/' + playlist["playlist_name"][0] + '_regression_coefficients.csv')

if __name__ == '__main__':
    AnalysisOne()
    AnalysisTwo()
```