1 Introduction

Background

Music preferences are highly variable across individuals and genres. Despite this variability, an overwhelming number of new artists and albums are released each year - as a result, our process of discovering new favorites can rely heavily on a few reviewers and “best-of” lists. With major music review sites wielding disproportionate influence on artist popularity, an important question is whether sites such as Pitchfork (“The Most Trusted Voice in Music”) are “unbiased” in their reviews. Namely, do their reviewers display bias towards certain genres or characteristics of music?

Perhaps, whether reviewers are “unbiased” is not even the right question to ask. Matching reviewer bias with consumer preference could provide more accurate music recommendations, a domain that streaming services such as Spotify already pose a threat in. There are now faster, more innovative ways for people to discover new music, and the Pitchforks of media must adapt to remain relevant and competitive in the increasingly fast-paced media landscape. Thus, they must experiment with new ideas, including methods of delivering music recommendations.

Project

In this project, I create a recommendation engine that matches consumers with compatible reviewers – consumers allow the Spotify API to access their top-played tracks, and the engine recommends music critics to follow, based on similar music preferences. The aim is to integrate the type of recommendation engine that has increasingly become the norm of music discovery, along with the enduring importance of “longform” reviews that formed the original basis of readership for these sites and their continuing appeal. Music critics are categorized by music preference to predict which ratings they will individually score for albums. These music preference “profiles” are then used matches individual consumers with compatible music critics (a “reviewer recommendation engine”).

2 Data Acquisition & Cleaning

Code: https://github.com/diana-xie/spotify_pitchfork_recommendations/blob/master/DataCleaning/data_wrangling.ipynb

In this project, I draw data from two sources, (1) Pitchfork SQL database of music reviews (1999-2017), (2) Spotify’s API.

Pitchfork

Pitchfork review data was extracted from a Pitchfork SQL database downloaded from Kaggle. The database contained all information relating to every review authored from 1999-2017, including author name, album, and rating. Once the data was situated in a dataframe within the Python environment, it was re-organized to prepare for analysis. Although the Pitchfork data

was already fairly cleaned, there remained a significant fraction of misspellings and duplicate authors. The bulk of data cleaning and wrangling focused on ensuring that reviewer names were consistent across the multiple reviews that they had authored.

Spotify

Spotify API data was cleaner, so the bulk of work focused on accessing Spotify's API and extracting the necessary information to produce feature analysis for each album reviewed in the Pitchfork database. Spotify has a specific category of metrics for album tracks called audio feature analysis (**Fig 1**), which is generated by a combination of proprietary machine listening techniques and the track's separate, more complex audio analysis of characteristics such as pitch, beats, and tatums.

We use the Python library Spotipy to access Spotify's API. To access Spotify's API (with or without Spotipy), we also had to register our project with Spotify as an "app" on their developer website, as this step is necessary to obtain secret keys to access Spotify's data.

acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic.
danceability	How suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.
duration_ms	The duration of the track.
energy	A measure from 0.0 to 1.0 representing perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale.
instrumentalness	Predicts whether a track contains no vocals. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
key	The key the track is in. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on.
liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
mode	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
speechiness	Detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words.
tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
time_signature	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
valence	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

Fig 1. Spotify's audio feature analysis – A list of each audio feature used in Spotify's track-by-track "audio feature analysis". https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/

Finally, since Spotify feature analysis is conducted on a per-track basis (i.e. not on an album as a whole), we had to determine a proxy for 'feature analysis' for an entire album. This proxy was to average the album's tracks' feature analyses, thus generating a set of features whose scores were the average of all its tracks' feature scores.

3 Data Exploration

3.1 Inspecting data for possible bias

Code: https://github.com/diana-xie/spotify_pitchfork_recommendations/blob/master/DataVisualization/data_story.ipynb

Pitchfork's data looks fairly clean and unbiased. In fact, there may have been deliberate steps taken to ensure that reviews were normally distributed **(Fig 2)**.

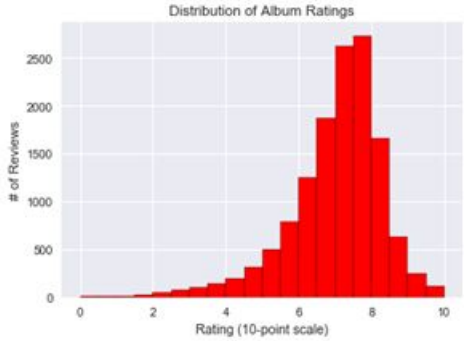


Fig 2: Distribution of album ratings. Normal distribution with slight left skew. A sign of lacking rating bias.

Reviews were also published on a consistent basis throughout the year (**Fig 3**), where years and ratings are not affected by the month of year the review was published (**Fig 4**).

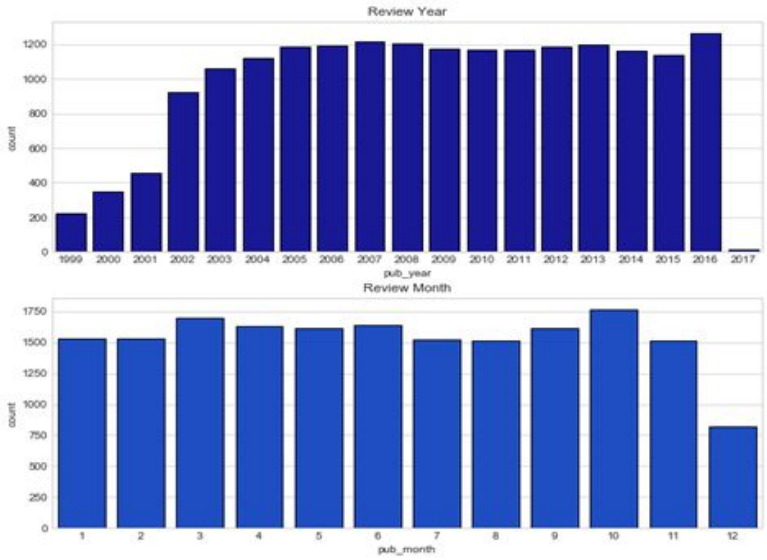


Fig 3: When do reviews tend to be published? When reviews are released/written. Generally appears to be little to no bias, with the exception of a noticeable lull in December when critics are probably on holidays.

Additionally, Pitchfork's team of music critics have a range of experiences reviewing for the site (**Fig 5)**, and their experience does not affect how high or low they rate albums.

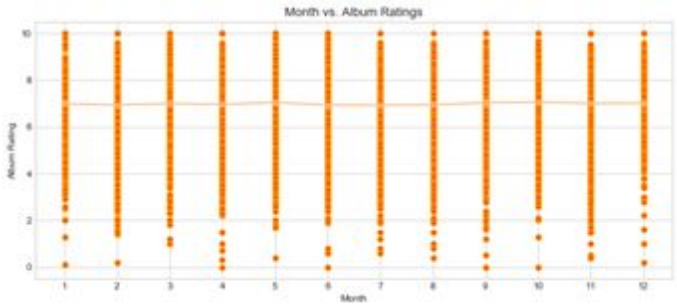


Fig 4. Rating vs. Month. Time series where review publication month is plotted against average album rating of the month, to check whether there is rating bias in month of the year.

Fig 5. Number of reviews written by critics. Examining critic experience - number of reviews they've written for Pitchfork.

The album's streaming popularity likely does not affect ratings either (**Fig 6**).

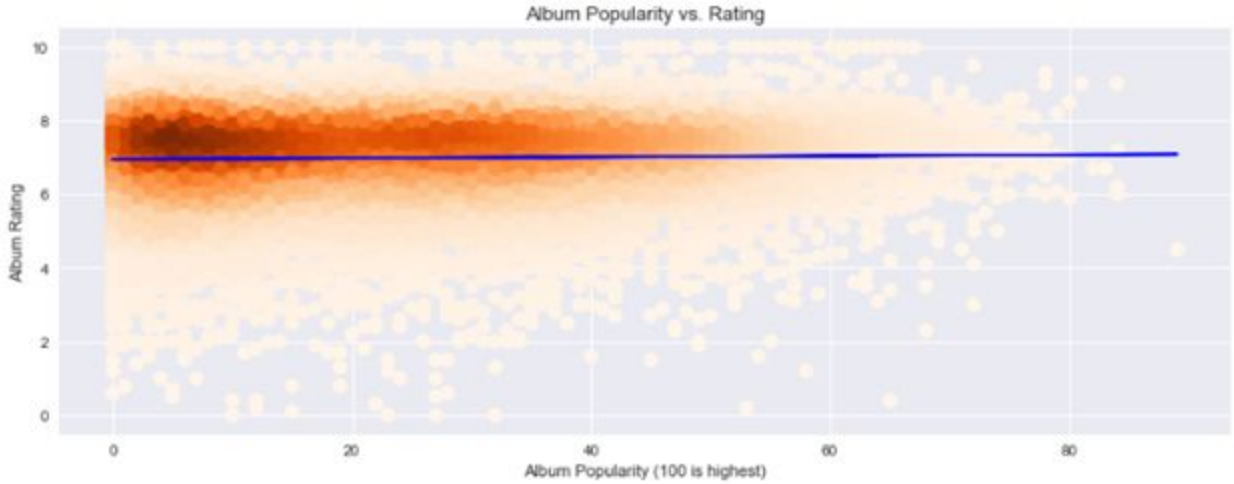


Fig 6. Album popularity (number of Spotify streams) vs. album rating awarded by Pitchfork critics. Linear regression, p-value = 0.011, which would fail alpha = 0.01 but pass alpha = 0.05. Weak correlation with Pearson's R = 0.02, slope = 0.0016.

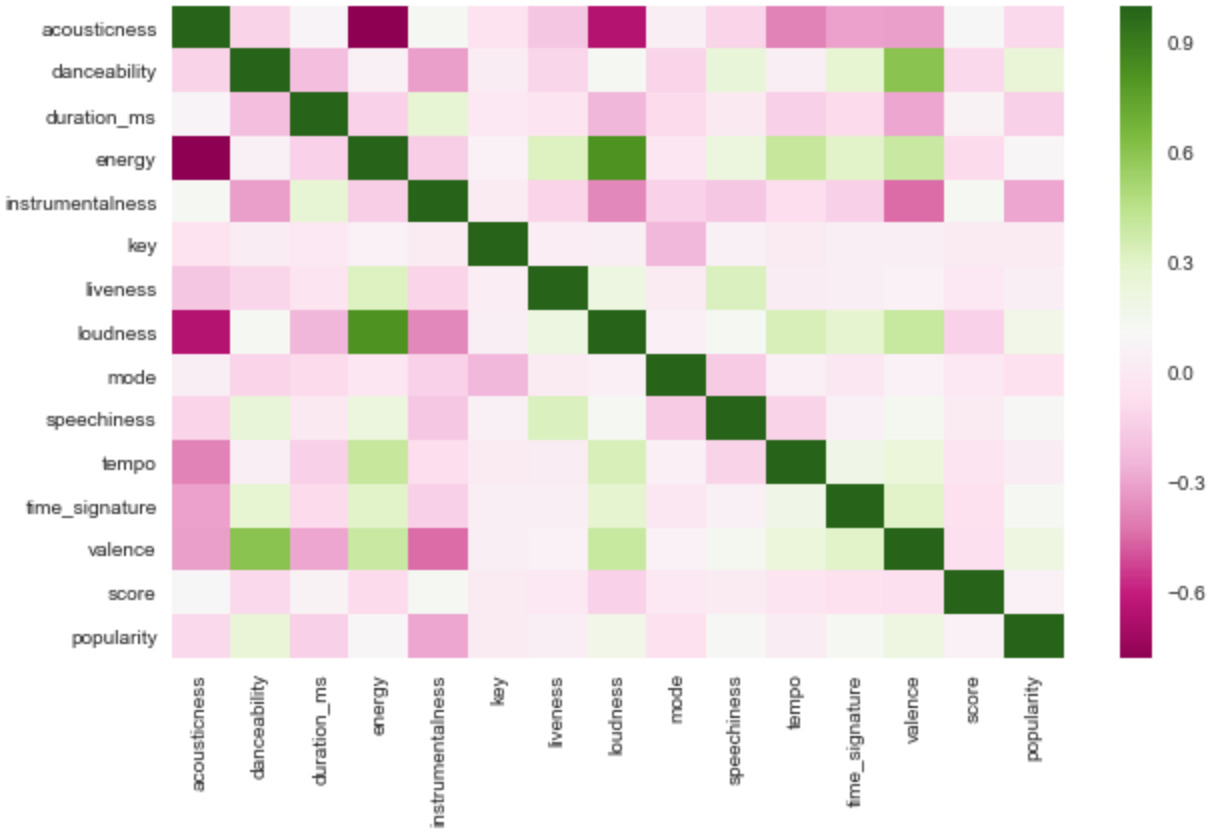
Unfortunately, album genre from Spotify features was not available for any of the albums reviewed. A useful relationship to investigate would have been the types of genres music critics tended to review.

3.2 Do audio features influence album ratings? (entire dataset)

Code: https://github.com/diana-xie/spotify_pitchfork_recommendations/blob/master/DataVisualization/data_story.ipynb

After inspecting the data for possible bias and trends, we turned our focus determining how to meaningfully group music critics for the recommendation engine.

First, we checked whether there were correlations between audio features and album rating from the entire dataset (**Fig 7**). Although there are strong correlations between a few audio features, there is weak correlation between any specific feature and album rating



.

Fig 7. Correlation matrix between audio features and album rating. Strong correlations between a few audio features, but weak correlation between any specific feature and album rating (“score”, boxed in red).

3.3 Discussion: Modelling individual music critics

Based on our correlation matrix, no audio feature alone is likely to bias album ratings. Therefore we proceeded to focus on analyzing critics on an individual basis. Perhaps it is not an individual feature, but a suite of features that vary from critic-to-critic, that predict album ratings.

To test this hypothesis, we must construct a predictive model for each critic, where the predicted output is album rating and regressors are audio features of the albums the critic rated, weighted by the score they awarded the album. The purpose of this modelling is to determine whether a critic preferred albums with certain audio feature characteristics (i.e. would be more likely to award a favorable review). A suite of predictive audio features would effectively serve as a musical taste profile for the critic. For instance, a critic may prefer albums with high danceability and instrumentalness scores.

3.4 Is linear regression suitable for grouping music critics?

We want to determine whether it is possible to assign an accurate individual model to each critic, which would allow us to group together critics with similarly weighted features (regressors). In this model, the regressors are album features and predicted outcome is the rating they assigned to the album.

To accomplish this task, we started with one randomly selected critic to test our idea. We used sklearn's linear regression to model a music critic on audio features of album vs. album's rating they awarded (**Fig 8**).

First, we normalized all the features, so that features with larger ranges of values did not distort the regression. Then, we tested how well such a linear regression model would perform on a randomly selected music critic.

We found that the individual critic model was insufficient to predict album ratings. The model had weak fit and correlation (Pearson's R = 0.13, p-value = 0.48).

Removing weak coefficients did not sufficiently improve the model either.

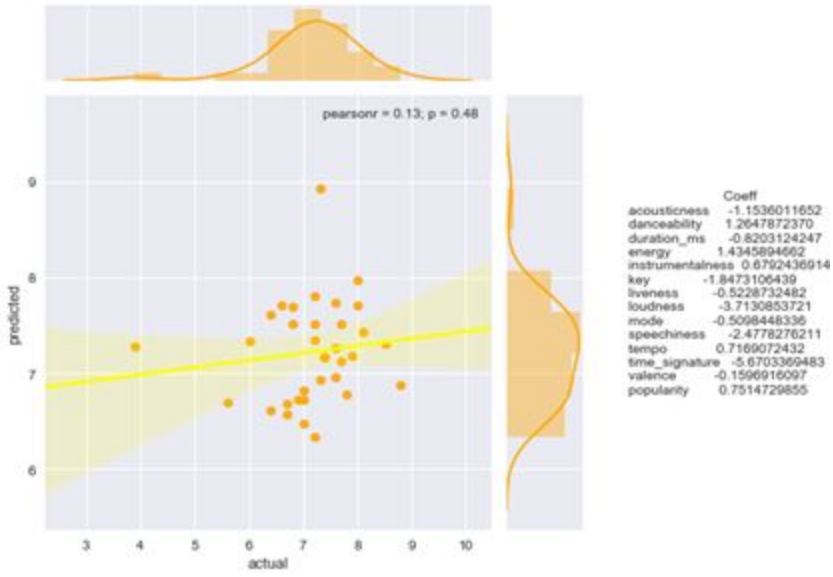


Fig 8. Actual vs. predicted album ratings for a music critic. The critic we chose, Aaron Leitko, authored 83 reviews - on the higher side of # reviews for a nice sample size.

3.5 Discussion: combining linear regression with clustering methods

We saw that implementing linear regression on an individual-critic basis was not sufficient to predict ratings. A likely factor was an insufficient sample size of reviews for each critic. Our sample reviewer had 83 ratings, on the higher end compared with other critics in our database. However, this was insufficient to produce a statistically significant model. Therefore, we would likely not expect statistical significance when performing linear regression modelling (on an individual-critic basis) for the majority of critics in our dataset.

A solution is to cluster critics first, then perform linear regression modelling on the pool of critics in each cluster – i.e. model each cluster with the same linear regression methods performed here. We would accept each "high quality" cluster that yields a pool of critics with a statistically significant model, where the model performs sufficiently well at predicting album ratings for the

critics in that cluster (based on their music preferences, i.e. weighted audio features). Accordingly, we would reject low quality clusters. This cluster selection could be performed in an iterative fashion, until all critics are assigned to a cluster (i.e. similar musical taste group).

In the next section, we describe our clustering data exploration to determine the ideal process for selecting and assigning clusters.

4 Cluster EDA: KMeans

Our ultimate goal is to set up an automated process for selecting clusters that group music critics appropriately by their preferred audio features. Here, we outline the steps leading to our setup.

To set a standard for comparison to other clustering methods, we began with KMeans clustering (sklearn) and subsequently tested cluster quality with our aforementioned linear regression modelling process (previous Section 3.5).

4.1 Weighting audio features by album rating

We use weighted audio features to cluster each critic. Specifically, we compiled album ratings for each critic and the audio feature scores associated with each album. These feature scores were then weighted by the rating the critic awarded the album – features were multiplied by the critic's album rating and normalized to ensure that critics whose relative rating scale was higher or lower than the norm would not appear skewed relative to other critics (**Fig 9**).

author_fullname	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode
aaron leitko	1.7669613509	2.7102818648	0.4394045530	4.7816769157	3.6834826070	3.4459148373	1.3192222425	5.6089777642	5.1194929396
abigail covington	1.4951118364	4.5360173469	0.3154418588	4.3567193344	0.0626842169	3.6649783550	1.3568244620	5.7701599203	3.6304761905
abigail garnett	1.1352992396	5.0830765529	0.4835500103	4.8579345464	4.7580254769	3.5848963663	0.7246622576	5.4236931736	4.3652669553
adam dlugacz	0.0012127985	2.0102734596	0.2351180050	5.2985298573	0.4313224214	2.4230769231	2.0638801090	5.4037899891	4.8461538462
adam moerder	1.2598394617	2.8502051162	0.2772205694	4.2148253480	1.5645504890	2.9592236849	0.9925679705	4.9584907427	4.1454142274

Fig 9. Data matrix input to clustering algorithms. Dataframe is indexed by author, where each row represents their normalized and weighted score for the audio feature (columns). (Dataframe truncated column-wise for space.)

4.2 Determine number of clusters to use

In order to determine cluster size and adjust other parameters for our KMeans clustering, we performed an elbow analysis.

The elbow method involves plotting SSE (sum squared errors) for progressively larger number of clusters (k) specified to the algorithm and then picking a k where SSE decreases abruptly. The idea is that SSE decreases as k gets larger, and at some point the marginal return of higher k is not worth increasing k further.

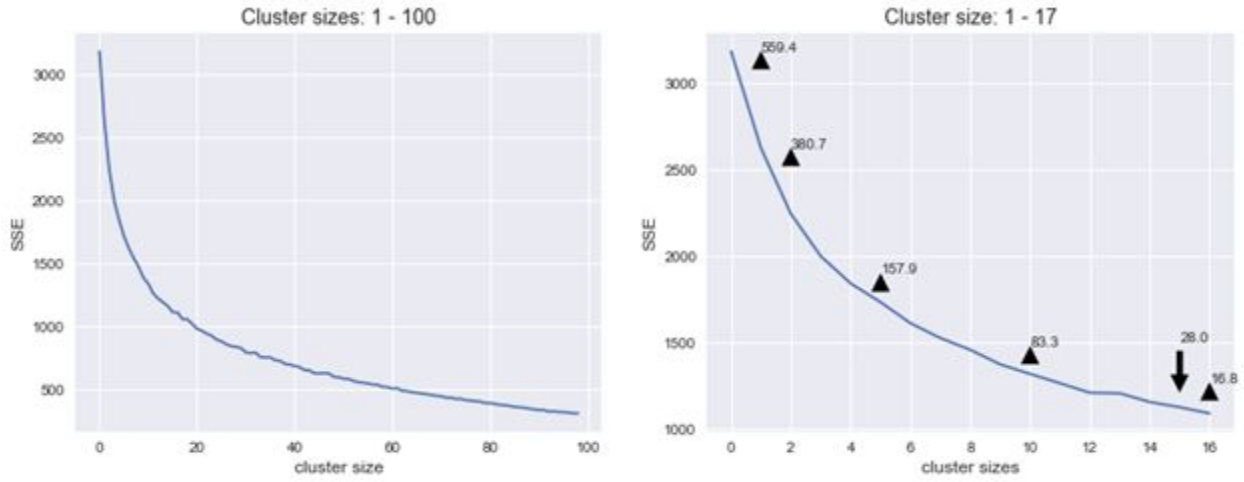


Fig 10. SSE vs. cluster size. Using the elbow method to plot SSE vs. different number of clusters.

By inspecting the resulting p-values, correlation metrics, and cluster sizes on each cluster, we were able to determine how well linear regression modelling performed on each cluster.

Our conclusion was that more than 12 clusters was not worth the tradeoff of more clusters, which would reduce the size of our train and test sets when performing cross-validation for clusters. In fact, greater than 12 clusters tended to cause an error in the clustering algorithm, where some clusters were empty.

	p_values	r_values	size_authors	size_reviews	slopes
0	8.4281635248e-01	0.0560188833	12	47	0.1007128055
1	1.3352678709e-12	0.3459129074	30	1322	0.1328197334
2	3.7534632605e-06	0.2567954016	23	1051	0.0766557291
3	6.8643326643e-01	0.0237642588	48	968	0.0043234448
4	1.3182442876e-08	0.2250554332	59	2078	0.0581557115
5	2.2779801305e-09	0.1786292608	68	3677	0.0418224226
6	2.5343484480e-01	0.3989504336	10	31	0.4496634750
7	3.8699379756e-01	0.3900774221	13	21	4.6867846822
8	3.5435412880e-05	0.1858526572	38	1628	0.0374079999
9	8.3707893142e-02	-0.9913678938	7	10	-1.0743266499
10	1.9205384104e-07	0.1889799852	54	2492	0.0453827468
11	0.0000000000e+00	1.0000000000	4	4	0.9038361540

Fig 11. Cluster quality metrics. Linear regression metrics on clusters generated from a KMeans iteration, where number of clusters (k) = 12.

As a result, we set our maximum k clusters limit to 12.

4.3 Criteria for selecting “high-quality” clusters

To set up an automated process for selecting clusters, we must specify a set of criteria for accepting and rejecting clusters. In doing so, we minimize case-by-case subjectivity and must consider what constitutes a “well-performing” cluster for purposes of forming our recommendation engine.

Using linear regression to check cluster “quality”

As mentioned in our earlier discussions of pairing linear regression with clusters (Section 3.5), we use linear regression modelling to test whether a cluster has captured similar music critics. The model is trained on 70% of the cluster’s music critics, and the remaining 30% are used to test how well the model predicts these test critics’ actual album ratings. How well the model fits the

data (weighted audio features vs. predicted album rating) and its statistical significance determines cluster quality.

Setting criteria for accepting "high-quality" clusters

Recall that our goal is to set up an automated process for selecting clusters that group music critics by their preferred music tastes. To do so, we will need to specify a threshold for clustering and linear regression metrics, above which we accept clusters and subsequently remove these clusters' assigned critics from the sample. This sample, with clustered critics removed, will then have clustering performed on it again so that all critics are eventually assigned to a cluster.

We eventually set upon the following criteria:

- **Number of reviews > 10.** There needs to be more than 10 reviews in the cluster, otherwise the cluster size is impractically small and our recommendation engine may rely on too many overfitted clusters.

- **Number of authors > 6.** Unlike reviews (13585 total), there are only 366 unique critics after wrangling the data. Since the clusters that our recommendation engine relies on will be based on similarity between critics, only at least 2 critics are required to create a cluster.

  However, this should be balanced with considerations about overfitting and whether a consumer or our held-out test set of critics (who were never part of the clustering sample) will receive accurate results when we attempt to cross-validate our recommendation engine.

- **Pearson's R > 0.5.** Determining a sufficient R-value requires us to consider the subjective territory of how many points difference a rating would make on a consumer's perception of the album. Pitchfork's album rating system is 0-10. Would a scale of 0.1 or 1 points make a difference? Any R < 0.5 would mean that the rating could be off by many points, which could make a drastic difference between whether an album is considered average or outstanding - even an 8.0 vs. 10.0-rating album can mean the difference between simply an above-average album or a note-worthy and outstanding album. We experimented with setting the R-value criteria higher. Although we could benefit more stringent R-values, it is nearly impossible to cluster many critics with anything higher than 0.5.

- **Cross-validation score > 0.7 and p-value < 0.05.** For standard statistical significance, we set our alpha to 0.05. Or cross-validation threshold was set to 0.7, below which would indicate that the model is not even generalizing to 30% of critics in its cluster.

5 Cluster EDA: Alternative clustering algorithms

Code: https://github.com/diana-xie/spotify_pitchfork_recommendations/blob/master/DataClustering/data_clustering.ipynb

While experimenting with different KMeans parameters, we also tested other clustering methods to determine whether KMeans was the ideal algorithm to group music critics.

We used silhouette scoring to compare alternative methods. Before accepting any clusters, we checked to see how well it performed under our linear regression modelling method for determining cluster quality (Section 3.5).

5.1 PCA: Does reducing data dimensionality produce better clustering?

Before proceeding with different clustering methods, we wanted to see whether reducing the dimensionality of our features would produce better KMeans clustering.

We reduced our data to 2 components for easy visualization and colored the resulting plot by cluster assignment (**Fig 12**). The results were promising - there was clear separation between the 3 clusters under this PCA.

However, when we proceeded to the next step of checking cluster quality with linear regression models, we found that each cluster did not perform as well as our KMeans clustering (k = 12 clusters). Our Kmeans clusters still had higher R-values, cross-validation scores, and slopes of regression (actual vs. predicted album ratings).

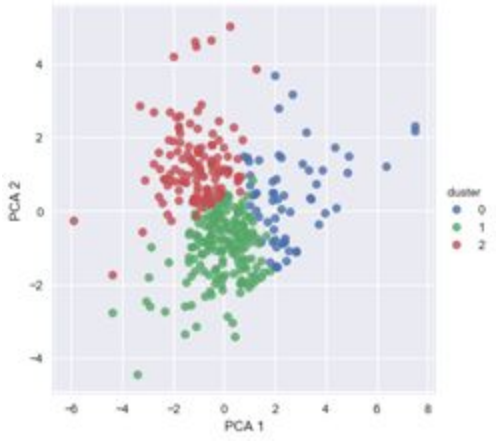


Fig 12. PCA. Plotting principal component (PCA) 1 vs. PCA 2.

Additionally, we found each principal component (PCA) difficult to interpret – which features distinguished one PCA from the other and how this translated into musical preferences was difficult to parse. Given these two obstacles, we therefore proceed to test other clustering methods without reducing the dimensionality of our data.

5.2 Results: Other clustering methods

We used silhouette scoring, which measures how well each data point fits its assigned cluster and how poorly it fits others.

The following scores were determined using default parameters for these clustering methods:

Range	Interpretation
0.71 - 1.0	A strong structure has been found.
0.51 - 0.7	A reasonable structure has been found.
0.26 - 0.5	The structure is weak and could be artificial.
< 0.25	No substantial structure has been found.

Fig 13. Criteria for silhouette coefficients. At 0, clusters overlap. -1 (poor clustering) to +1 (very dense clustering)

- **Affinity propagation** - Average silhouette_score: 0.10679604704138726
- **Spectral clustering** - Average silhouette_score: -0.17307529392989118
- **Agglomerative clustering** - Average silhouette_score: 0.10522049997344707
- **DBSCAN** - Average silhouette_score: no score, since only 1 cluster generated

None of these methods yield particularly good silhouette scores, although spectral clustering comes close to our KMeans silhouette scores generated earlier. We continue with our original KMeans algorithm.

6 Recommendation engine

Our process of discovering “similarity” between the user or one of our held-out critics and the remaining sample of critics consists of the following steps:

1. From each critic in our table, we take the difference between their average features and the user's to effectively make a matrix of differences, where index = critic and column = (critic averaged feature) - (user averaged features).
2. *(This step only for Spotify users, not held-out critics)* Then, for each feature column we scale these values (differences) to keep larger differences in our larger-scaled features from distorting our distance metric.
3. For each critic, we average their difference values.
4. Take the critic with the least difference from user as the ***first-choice critic***.
5. Identify this critic's cluster and include remaining critics in cluster as ***second-choice critics***.
6. Backup method: If critic’s first-choice critic does not belong to a cluster, in lieu of #5’s “first-choice critic → first-choice critic’s cluster” approach, we simply take the cluster with closest averaged features.

This process effectively takes the Manhattan distance between the recommendation engine user (or held-out test set of critics) and the critics that constitute our clustering sample. We were careful to use only normalized features, as some features are inherently on larger scales and could thus distort difference values after averaging.

7 Cluster cross-validation

7.1 Process & Results

Finally, we use use critics held-out from the clustering sample to input to our recommendation engine to evaluate how well their album ratings are predicted by the cluster of music critics they are recommended.

Before clustering on our sample, we randomly removed 20% of the music critics from the sample as the “cluster cross-validation” holdout set. Then, we input them into the recommendation engine as users (test critics) and matched them with a first-choice critic and the cluster belonging to the matched first-choice critic.

Finally, we performed linear regression modelling on the cluster and cross-validated the test critic’s album ratings to those predicted by this model. A sample of our results are below:

	cluster_num	first_choice	p_value	r_value	score	slope
0	5	no	3.3075691844e-04	0.5793035797	0.3355926375	0.3355926375
1	5	no	3.3075691844e-04	0.5793035797	0.3355926375	0.3355926375
2	2	no	1.9302210231e-04	0.6210228739	0.3856694099	0.3856694099
3	2	no	1.9302210231e-04	0.6210228739	0.3856694099	0.3856694099
4	5	no	3.3075691844e-04	0.5793035797	0.3355926375	0.3355926375
			.			
			.			
28	5	yes	3.3075691844e-04	0.5793035797	0.3355926375	0.3355926375
29	2	no	1.9302210231e-04	0.6210228739	0.3856694099	0.3856694099
30	3	no	6.3069610067e-04	0.5881345496	0.3459022485	0.3459022485
31	1	no	2.2996417163e-06	0.7123145829	0.5073920650	0.5073920650
32	5	no	3.3075691844e-04	0.5793035797	0.3355926375	0.3355926375
33	2	no	1.9302210231e-04	0.6210228739	0.3856694099	0.3856694099
34	0	no	5.8074333531e-07	0.5429219567	0.2947642510	0.2947642510
35	5	no	3.3075691844e-04	0.5793035797	0.3355926375	0.3355926375
36	1	no	2.2996417163e-06	0.7123145829	0.5073920650	0.5073920650
37	5	yes	3.3075691844e-04	0.5793035797	0.3355926375	0.3355926375

Fig 14. Cross-validation results. Each row represents a held-out critic and the results given to them by the recommendation engine.

Key: **cluster_num**: their matched cluster; **first_choice**: whether their first-choice critic match belonged to a cluster; **p_value**: p-value of linear regression model; **r_value**: Pearson's R of linreg model; **score**: cross-validation score (held-out critic vs. the matched cluster); **slope**: slope of regression (linreg model)

Cross-validation scores ranged from ~0.29 to 0.98. In the cases where cross-validation (CV) scores were extremely poor, the cluster matches (i.e. second-choice recommendations) were generated by the backup method of identifying best-match cluster in lieu of the first-choice critic belonging to a cluster. Overall, the majority of CV scores were below 0.5, with an average of 0.39. Although we would have preferred lower CV scores, our scores indicate that our recommendation engine is somewhat good at recommending music critics, and its usefulness generalizes to random users.

The cross-validation scores of these cluster matches are similar to what we find if we only accept first-choice critic's clusters (first_choice = yes vs. no). Therefore we accept our method of taking most similar cluster as proxy for second-choice music critic recommendations, in cases where the first-choice critic was not assigned to a cluster.

7.2 Discussion

First choice critic + second-choice critics. Since only 61 of our critics ended up being clustered, the likelihood that our held-out critic would be matched to a clustered first-choice critic is fairly low. To address this issue, if the held-out critic was matched to a first-choice critic who didn't belong to a cluster, we found whichever cluster had the most similar overall averaged features.

As a result, there were two categories of outputs from our recommendation engine:

1. Accurate first-choice critic + reliable second-choice critics, or

2. Accurate first-choice critic + less reliable second-choice critics

It was therefore useful to frame our recommendations as pairing the consumer with their most similar critic (the first-choice critic) and then presenting the remainder of music critic recommendations as "second-choice" critics.

A few other notes:

- **A few critics are incredibly similar, but the majority are not**. Over the course of our experimentation with different clustering parameters and iterations, we noticed a few clusters that had strong correlation and cross-validation scores among the critics belonging to that cluster. However, the remaining unclustered critics (due to lack of similarity) remained in the majority.
- **Good clusters, but not enough of them.** Although finding a few clusters with strong within-cluster musical agreement among its critics is useful for identifying music critics with similar tastes, for purposes of our recommendation engine the number of clusters was impractically low.

We proceeded with creating our recommendation engine. This recommendation engine outputs the user's top, first-choice critic and then their secondary matches (which are less likely to fit the user's tastes, depending on whether the top critic belonged to a cluster).

8 Final recommendation engine

Our final step is to match customers (users) with music critics.

Initially, we proposed taking user album ratings to match them with critics - this makes sense, given that album ratings are our basis for clustering critics. However, we opted instead to get user data from their Spotify “most-played tracks” data for several reasons:

- Asking users to rate albums on the spot is time-consuming and subjective - likely that a user's ratings will change with each use-case of the recommendation engine.

- Most-played tracks have the same audio features as album features

- Grabbing for example, the top 50 most-played tracks, is more likely to introduce similar diversity to the user's rating repertoire as a Pitchfork critic rating multiple albums

In summary, we use the user’s Spotify data -- their “most played tracks” -- to match them with compatible music critics.

8.1 Get user's top tracks/Spotify features

We grab the top 50 most-played tracks from the "medium-term" time range. The number of tracks and time range can be changed readily in the code. In this example, I use my username,

but any user can execute the code, be queried to login via Spotify, and give permission for the code to grab their top tracks data.

```
range: medium_term
0 Liebestraume No. 3 In A-flat Major (Nottorno No. 3) // Jean-Yves Thibaudet
1 Truant // Burial
2 Hyper Object // Blank Banshee
3 Teen Pregnancy // Blank Banshee
4 The Low Places // Jon Hopkins
5 Une Barque Sur L'océan from Miroirs // André Laplante
6 passionfruit // yaeji
7 Meteor // the bird and the bee
8 Futile Devices (Doveman Remix) // Sufjan Stevens
9 Mystery of Love // Sufjan Stevens
10 Love Is Stronger Than Pride // Amber Mark
11 Drifting Away // biosphere
12 Deep Space // Blank Banshee
13 Panacea // Disasterpeace
14 My Machine // Blank Banshee
15 Wheel // Visible Cloaks
16 Shadow // Chromatics
17 On & On // Cartoon
18 My Love // the bird and the bee
19 B:/ Start Up // Blank Banshee
20 Neon Pattern Drum // Jon Hopkins
21 C O S M // Jon Hopkins
22 M.A.Y. In the Backyard // Ryuichi Sakamoto
23 Nine // Autechre
24 Mine // Bazzi
25 Oh my my // Smerz
26 Germination // Ryuichi Sakamoto
27 Guap // yaeji
28 Pause // Harris Cole
29 Warm In The Winter // Glass Candy
30 Never Get To You // Moon Boots
31 We Disappear // Jon Hopkins
32 4:00 AM // Desired
33 Someone Else's Mantra // Club Kuru
34 The Evil That Never Arrived // Stars Of The Lid
```

Fig 15: Code returns list of 'top 50' most-played Spotify tracks (partial screenshot)

Then we compile Spotify's audio feature analysis for each track and average.

Example output:

```
Your top match is: barry walters
Your other matches are: ['jesse jarnow', 'jonathan bernstein', 'stephen erlewine', 'michael agovino', 'rohan samarth', 'john ev
erhart', 'stephen duesner', 'stephen may', 'philip shelley', 'christopher drabick', 'daniel crumb']
```

9 Conclusion

In this project, we wrangled Pitchfork and Spotify data to cluster all music critics who reviewed an album for Pitchfork between 1999-2017. We were able to accomplish this by integrating their album ratings with Spotify-provided album audio features.

Although only a handful of useful clusters were generated, we were still able to set up an automated process for picking tight clusters and combining linear regression as a form of

cross-validation to check that they were of practical use and actually incorporated critics with similar musical preferences.

In the end, we revised our recommendation engine to first recommend the user's most-similar critic (first-choice critic) and then a list of secondary recommendations that contained the remaining critics in the first-choice critic's cluster. Our approach was to prioritize recommending the *most* similar critic to the user, rather than having the usefulness of our recommendation be washed out by simply recommending an entire list (cluster) of critics that likely did not generalize as well (based on our cross-validation findings on critics held-out from the cluster sample).

In conclusion, we were able to set up a pipeline for integrating static data (Pitchfork SQL review/critic data downloaded from Kaggle) with that of an API (Spotify). We used this information to build a unique recommendation engine that recommended music critics, rather than the typical music track recommendation engine. Although more data is likely needed to refine the accuracy of our recommendation engine, it was still able to perform well in a few cases and in the least provided at least 1 reliable music critic recommendation.