# A Spotify-Pitchfork integrated recommendation engine

Diana Xie

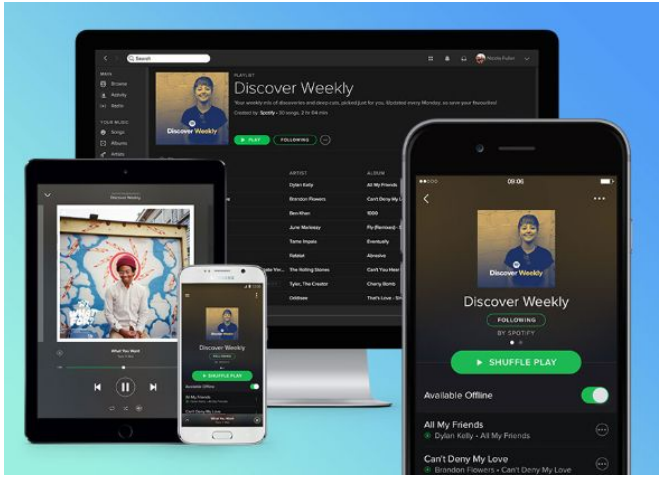
Data Science Career Track Project, April 2nd 2018 Cohort

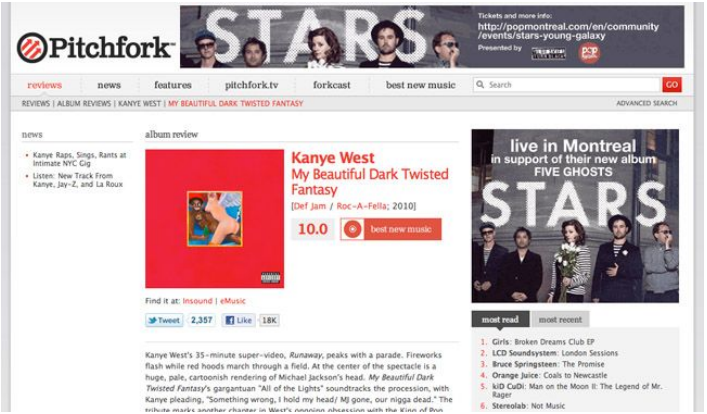
*Special thanks to my Springboard mentor:*

**Tommy Blanchard**
Data Science Lead,
Fresenius Medical Care

# The problem

- Music journalism and music discovery
  - The rise of music recommendation engines (i.e. Spotify)
- Music critic bias - not necessarily a bad thing!
  - Who is your best music critic match?
- A need for innovation for traditional media review sites (i.e. Pitchfork)
  - How to balance quality music journalism w/ the fast-paced media landscape?



# Who might care?

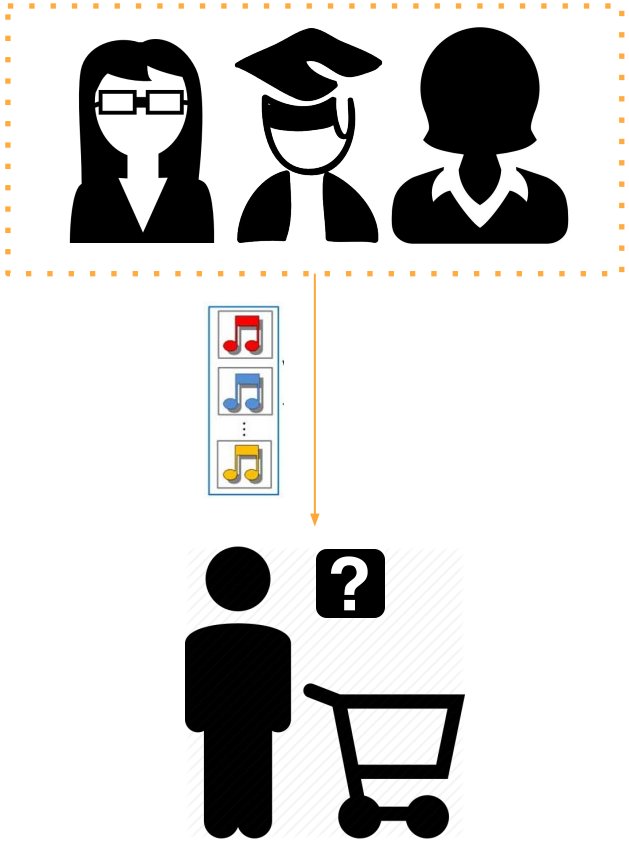
- Media sites specializing in media reviews (rating scale) authored by experts
    - Platforms where “longform” reviews formed the original basis of readership/continuing appeal.
- Companies interested in producing an app/platform that utilizes a “compatibility” engine
    - A recommendation engine that matches consumer preferences with compatible experts




# Proposal

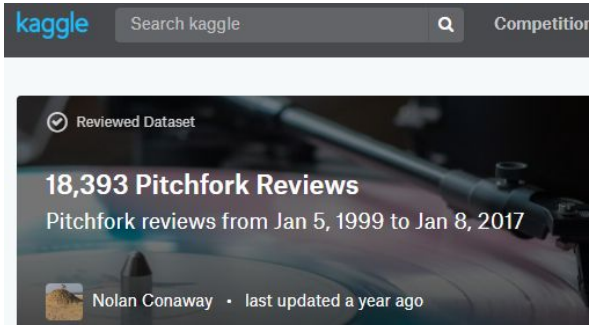
Create a “music critic” recommendation engine – consumers input preferences, are returned “compatible” music critics to follow

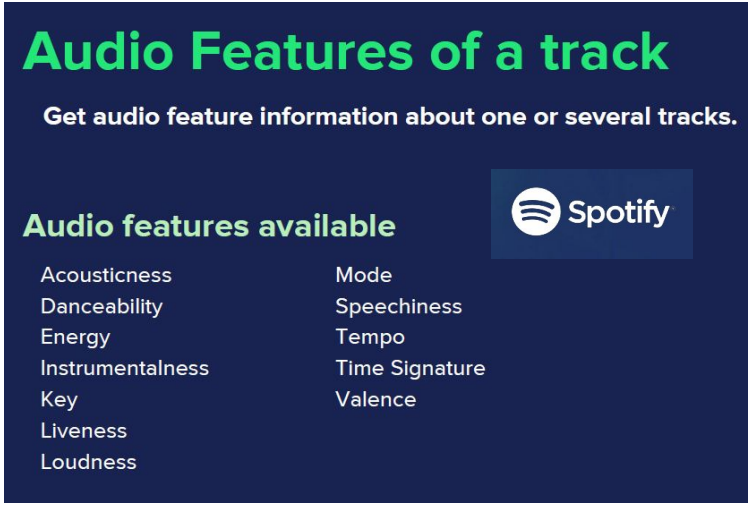
1. **Group** music critics by music preference
2. **Predict** what ratings they will individually score for albums
3. **Generate** music preference “profiles” - used to match individual consumers with compatible music critics



# Data sources

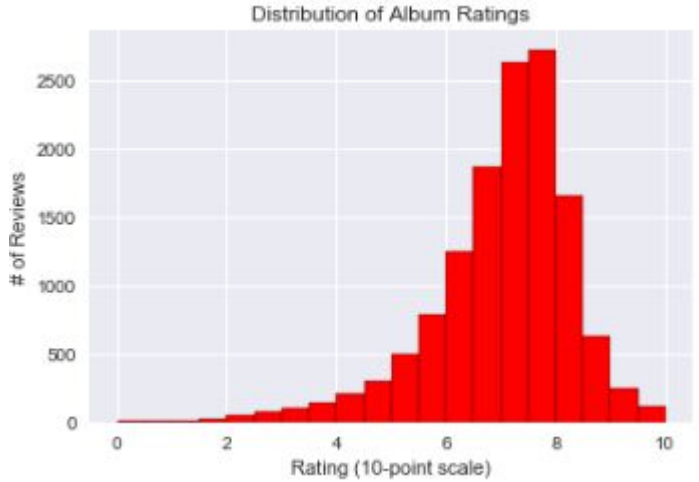
- Pitchfork SQL database of music reviews (1999-2017)
  - ~ 500 unique music critics
  - ~ 18400 album ratings
- Spotify's API
  - Audio features
  - Albums, Artists, Tracks, and more
  - Spotipy – a Python library to access Spotify API



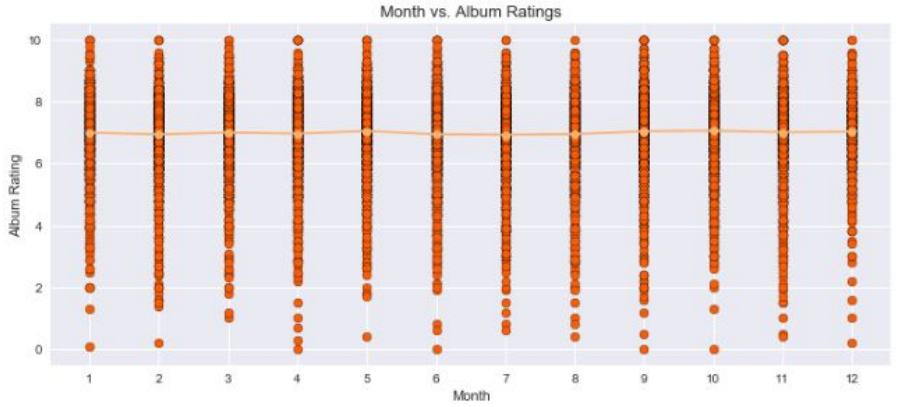



# Are album reviews biased?

- Normally distributed album ratings
- Album ratings consistent over time
- Critic review experience is diverse



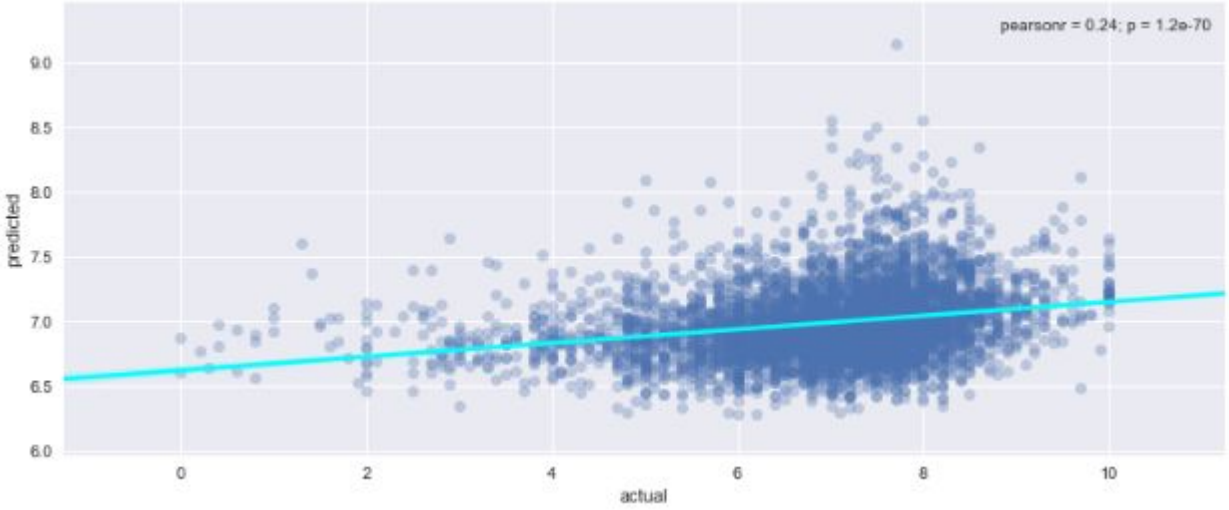


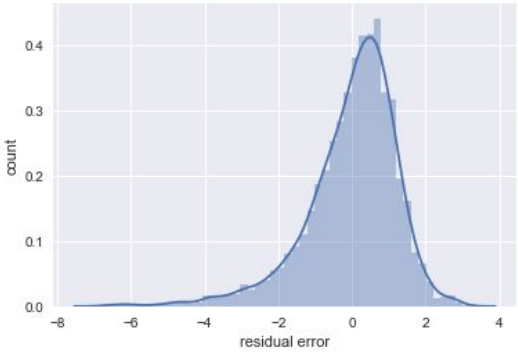


# Can one model generalize across all critics?

- Many of the coefficients are statistically significant and actually pretty impactful
- However, discrepancy between predicted & actual ratings too high to be practical

Linear regression model results: Actual album score vs. predicted score



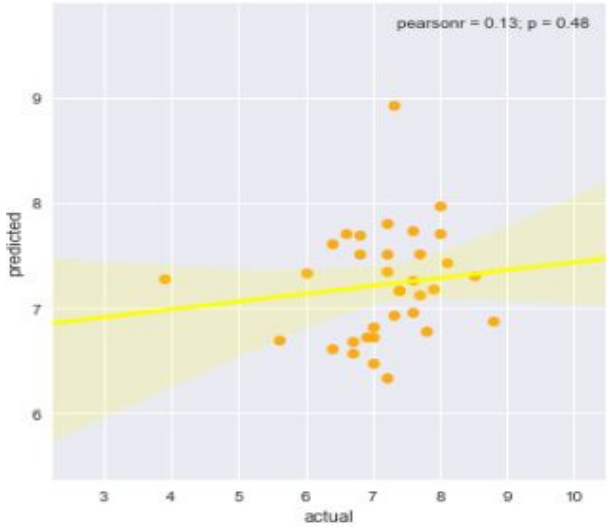



# Can we build individual models for critics?

The individual critic model was also insufficient to predict album ratings

- Poor fit and correlation
- Removing weak coefficients did not sufficiently improve the model either
- Sample sizes (# of reviews) simply too small – need to pool together similar critics and *then* perform linear regression modelling

Actual vs. predicted album ratings for a music critic.




….therefore, since implementing linear regression on an individual-critic basis was not sufficient to predict ratings, we proceed with cluster EDA

# How to cluster music critics?

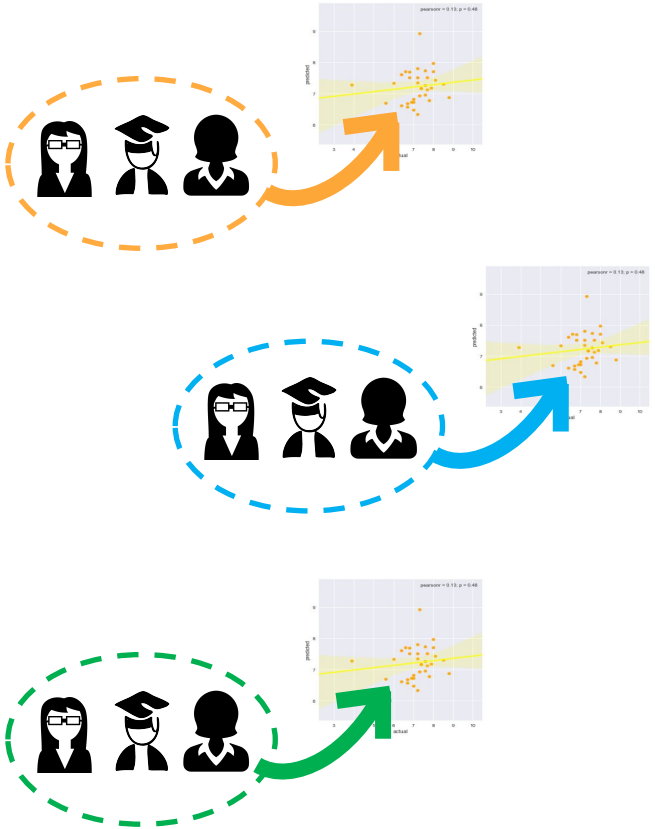
**Goal**: Pool together similar music critics (i.e. clustering) and generate a model for each pool.

**Method:** We use kmeans (sklearn) to cluster critics by weighed features

- Normalize features, then weigh (multiply) by critic's rating

**Test:** Now that similar critics are pooled together, how well does each pool's model predict actual album ratings for their cohort?

- Same linear regression model as before



# Setting criteria for “high-quality” clusters

Goal: Automate process of selecting clusters that group together similar music critics.

Approach:

1. Use linear regression modelling on each cluster to determine cluster “quality.

2. Only accept clusters that pass a set of cluster criteria:

    - **Number of reviews > 10.** Sufficient sample size of reviews in cluster.
    - **Number of authors > 6.** Sufficient sample size of critics in cluster.
    - **Pearson’s R > 0.5.** A sufficient R-value that allows us to achieve multiple clusters w/o sacrificing accuracy (predicted vs. actual album ratings)
    - **Cross-validation score > 0.7 and p-value < 0.05.** For standard statistical significance and generalizability of cluster.

| | p_values | r_values | size_authors | size_reviews | slopes |
|---|---|---|---|---|---|
| 0 | 8.4281835248e-01 | 0.0560188833 | 12 | 47 | 0.1007128055 |
| 1 | 1.3352878709e-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 |

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

# Recommendation engine: two versions

1. **Cross-validation version**

- **Consumers** - held-out critics (20% of music critics)
- **Recommendation engine input** - album ratings (Pitchfork)

2. **Commercial version**

- **Consumers** - actual users looking to receive music critic recommendations
- **Recommendation engine input** - their Spotify 50 “top-played” tracks from the medium-term range

**Rationale**:

- Asking users to rate albums on the spot is time-consuming, subjective, and will vary between uses
- Most-played tracks have the same audio features as album features
- Grabbing top 50 tracks more likely to introduce similar diversity as a Pitchfork critic rating multiple albums

# Recommendation engine: the process

1. **Feature differences (user vs. critic)**. Take difference between user's features and each critic's corresponding features, to make a difference matrix; index = critic, columns = (critic averaged feature) - (user averaged features)

2. **Scale differences.** *(This step only for Spotify users, not held-out critics)* To prevent larger-scaled features from distorting our distance metric.

3. **Average differences (each critic).** For each critic, we average their difference values = 1 overall difference metric.

4. **First-choice critic.** Take the critic with the least difference from user as the ***first-choice critic***.

5. **Second-choice critics.** Identify this critic's cluster and include remaining critics in cluster as ***second-choice critics***.

- If first-choice critic doesn't belong to cluster, match user with cluster w/ most similiar 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.

# Cross-validation results

- Cross-validation (CV) scores ranged from ~0.29 to 0.98. Majority < 0.50.
- Average CV score: 0.39

**Conclusion:**

Recommendation engine is somewhat good at recommending music critics, and its usefulness generalizes to random users

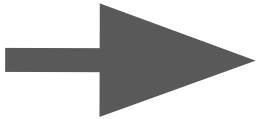
| | cluster_num | first_choice | p_value | r_value | score | slope |
|---|---|---|---|---|---|---|
| 0 | 5 | no | 3.3075691844e-04 | 0.5793035797 | 0.3355926375 | 0.3355926375 |
| 1 | 5 | no | 3.3075691844e-04 | 0.5793035797 | 0.3355926375 | 0.3355926375 |
| 2 | 2 | no | 1.9302210231e-04 | 0.6210228739 | 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 |

**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)

# Example run: consumer-side

```
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
```



```
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']
```

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

# Conclusion

- Set up a pipeline for integrating static data (Pitchfork SQL review/critic data downloaded from Kaggle) with that of an API (Spotify)
- Set up an automated process for picking high-quality clusters, using linear regression as a check for cluster “quality”
- Built 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.

# Thank you!