In this report I'll cover the steps to building a beer recommendation system.
The recommendation system built with user data from BeerAdvocate that can be found at https://data.world/socialmediadata/beeradvocate. The data (beer_reviews) is in a .csv format consisting of 1,586,614 observations and 13 features: brewery_id, brewery_name, review_time, review_overall, review_aroma, review_appearance, review_profilename, beer_style, review_palate, review_taste, beer_name. It contains over 10 years of data leading up to November 2010. Additional data is from https://www.beeradvocate.com/beer/styles/. This additional data (beer_description) included 108 style observations and 13 features: style, description, abv_high, abv_low, ibu_high, ibu_low.
The steps undertaken include: cleaning, visualization, logistic regression analysis, simple rankings, content (style) recommendations, collaborative filtering (by user ratings), evaluation and similarity metric trials.

Cleaning and visualization of data.
Cleaning consisted of three stages, correcting bad names in the review_profilename and brewery columns, correcting bad values in the beer_abv column, and lastly working with the beer styles. In all cases the values were encoded as nan but weren't identifiable as a string object or Numpy NaN value. To deal with this issue these columns were coerced to strings and then filtered by 'nan'.
Once filtered, the ratings for individual beers varied greatly by 'nan' review_profilename entries. For this reason all 'nan' profile names were changed to UNKNOWN. This was done so during profile ratings analysis there is an easy way to filter out these values.
After filtering brewery names by 'nan', there were two unique brewery ID's that contained bad entries (27 and 1193). Looking into beeradvocate with these identifiers yields no results. The next technique involved the individual beer names. If the beer names were unique to a specific brewery then the bad brewery names could be mapped based on the beers.
The beer names under the 27 brewery ID were not uniquely identifiable. Instead, they could be mapped to multiple breweries. Similar to review_profilename we changed these brewery names to UNKNOWN. The 1193 brewery ID contained additional information under beer name:

```
array(['Engel Tyrolian Bräu WRONG BREWERY SEE SCHWABISCH GMUND',
       'Engel Bock Dunkel WRONG BREWERY SEE CRAILSHEIMER',
       'Engel Gold WRONG BREWERY SEE CRAILSHEIMER',
       'Engel Landbier WRONG BREWERY SEE CRAILSHEIMER',
       'Engel Keller Hell WRONG BREWERY SEE CRAILSHEIMER',
       'Engel Aloisius - WRONG BREWERY SEE CRAILSHEIMER',
       'Engel Keller Dunkel  WRONG BREWERY SEE CRAILSHEIMER'], dtype=object)
```

Using the notes from this section the brewery names were mapped accordingly.
One major issue was found during this cleaning process. There are occasions where a brewery has multiple brewery entries under slightly different names. Some entries vary by as little as the use of an ampersand (&) or an and. No current analysis takes into consideration the breweries. This issue won't be addressed in this notebook for that reason.

The second stage of cleaning is working with the beer_abv column. Similar to before the bad entries are mapped under ‘nan’. These values are coerced to strings and then filtered. The first means of ameliorating these values was to find the set of beers that had ‘nan’ values and find their intersection with beers that have valid entries. Anything found at the intersection will then values mapped appropriately. This attempt failed due to multiple beers having non-unique names, names that are identical but from different breweries. The next option involved using the beer_description data.. Included in this data is Alcohol By Volume (ABV) ranges for each style. Using this data I took the mean of the ranges and mapped the bad ABV values based on style.

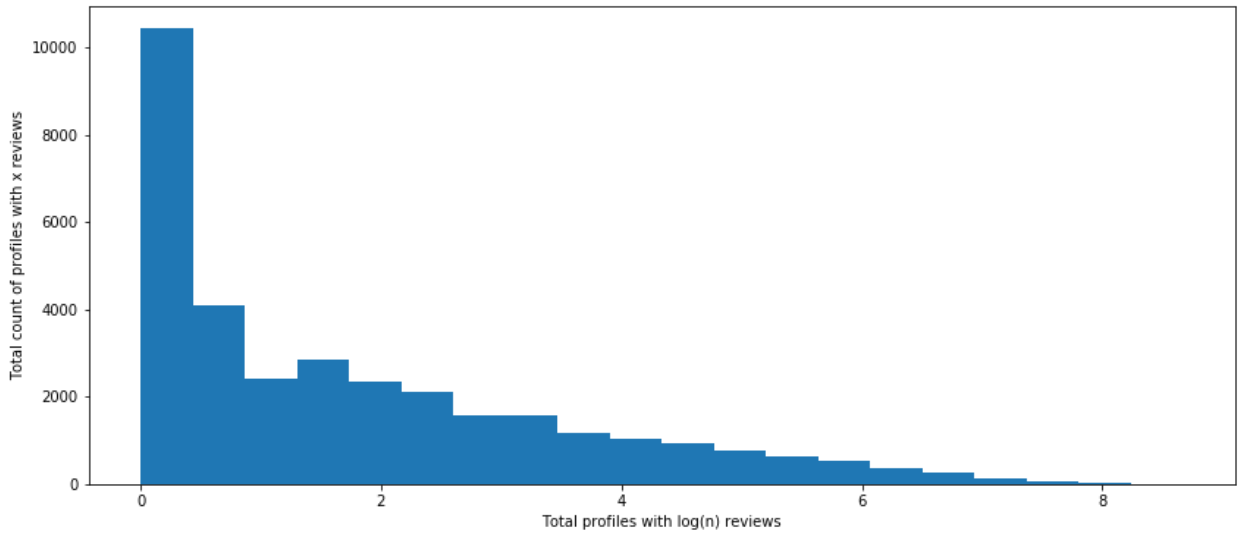
The final state of cleaning was to addressed differences in style naming conventions from beer_reviews and beer_descriptions. This data will be used across both data sets and are required to match.
Firstly I found at the intersection of styles between the two data sets. These are styles found in both data sets. Next I identified the styles that are different between the two data sets. There were 54 total styles that required correction. The naming schema used was from the beer_description data set. The styles from this data set are similar to the beer_review data, but more descriptive of the style, 'Belgian Tripel' instead of ‘Triplel’, or 'Russian Kvass' instead of 'Kvass'. To ensure quality matches between the two sets I found the correct style from the description data set and mapped it to the beer_review data. I then ensured that every style in the rating dataset had a match with the description set.
Finally I looked at any styles that were found only within the description set. These styles are new from the beer_review data set. To ensure the recommendations don’t suggest one of these styles during a content recommendation I dropped these styles. They were as follows:
'American Brett', 'American Brut IPA', 'American Imperial Porter', 'American Imperial Red Ale', 'Belgian Blonde Ale', 'New England IPA', 'Robust Porter', 'Smoke Porter'.
The argument could be raised that within the ratings dataset we had a beer classified as 'Smoke Beer' in the rating data set but would be classified as a 'Smoke Porter' instead. For simplicity sake I ignored this possibility. The alternative would be to identify the improperly assigned beers. Considering individual beers may have multiple unique beers under a single name this task seemed difficult and unnecessary.

Visualization

Visualization of the data included looking at the review ratings based on appearance, aroma, taste, palate, and overall. Each column was graphed as a histogram and Cumulative Distribution Function (CDF). The purpose of this was to visualize if there were values that fell out of the 0-5 range, therefore binning bias could be ignored. The result from this showed that all the values did fall between the 0-5 range.

This was done also to beer_abv to look for any outliers and negative values. No negative values were found, and the outliers that were present (beer ABV above ~15%) were found to be accurate for the beer.

Distributions (CDFs) of the data were built during the logistic regression model building as a means to look at linear relationships. These will be shown later in this report. Additional histograms were built using log scale to visualize long-tailed distributions such as:



Simple recommendations

Simple recommendation engines, or simple ranking algorithms, involve taking some item and using a metric to rank them from highest to lowest. This was done to three different items for the beer data set, brewery, style, and beers. All items were grouped by beer ratings using an aggregate of mean for the individual items. This created three separate datasets that included the unique items (either unique breweries, beers, or styles) and columns containing the mean of their ratings under appearance, aroma, taste, palate, and overall. After this was done the rankings were built by using the overall rating data and weighted based on count of total ratings. The formula used was IMDB's weighted rating system:

weighted rating (WR)=(v/(v+m))R+(m/(v+m))C
where:
R = average for the item (mean) = (Rating)
v = number of votes for the item = (votes)
m = minimum votes required to be listed
C = the mean vote across the whole report

In the simple recommendation the minimum (m) required varied depending on the item:

Style : 241	Brewery : 14	Beer : 3

The mean rating (C) across the items varied as such:

Style : 3.70	Brewery : 3.48	Beer : 3.61

Later implementations were built to include an average of all the rating characteristics. This was done to help ameliorate any possibly accidental entries for example:

Appearance	Aroma	Taste	Palate	Overall	Averaged
4	4	5	4	2	3.8
2	2	1	2	4	2.2

After the ratings were weighted they were sorted in a descending order. This descending order provided the simple recommendation method. Styles, beers, or breweries would be recommended in a highest to lowest system.

Beer Style Simple Recommendation (top 5):

	beer_style	review_overall	review_taste	review_appearance	review_palate	review_aroma	count	averaged	weighted_averaged
12	American Imperial Stout	4.029820	4.187230	4.163633	4.098669	4.160665	50705	4.145715	4.143621
32	Belgian Quadrupel (Quad)	4.071630	4.210909	4.117964	4.124986	4.132533	18086	4.148789	4.142927
94	Russian Imperial Stout	4.023084	4.149569	4.210072	4.086922	4.076576	54129	4.121774	4.119918
22	American Wild Ale	4.093262	4.149938	4.005451	4.040632	4.126756	17794	4.105069	4.099696
28	Belgian Gueuze	4.086287	4.127143	4.034864	4.046680	4.117574	6009	4.098602	4.083346

Brewery Simple Recommendation (top 5):

Individual Beer Simple Recommendation (top 5):
Before averaging:

	beer_name	review_overall	review_taste	review_appearance	review_palate	review_aroma	total_reviews	weighted_overall
41268	Rare D.O.S.	4.848485	4.848485	4.469697	4.803030	4.757576	33	4.748997
15244	Dirty Horse	4.820513	4.743590	4.423077	4.576923	4.615385	39	4.737236
47091	Southampton Berliner Weisse	4.768293	4.560976	4.182927	4.390244	4.353659	41	4.692361
3109	Armand'4 Oude Geuze Lente (Spring)	4.730769	4.730769	4.523077	4.669231	4.715385	65	4.683293
31374	M Belgian-Style Barleywine	4.750000	4.857143	4.482143	4.803571	4.785714	28	4.643997

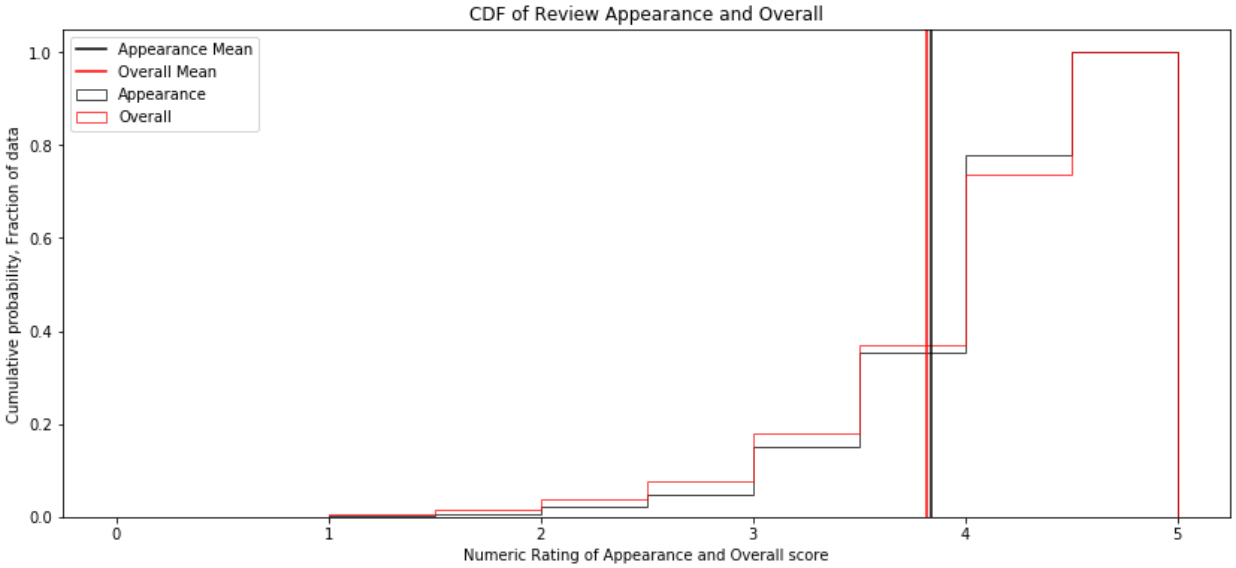
After averaging:

	beer_name	review_overall	review_taste	review_appearance	review_palate	review_aroma	total_reviews	averaged	weighted_averaged
41268	Rare D.O.S.	4.848485	4.848485	4.469697	4.803030	4.757576	33	4.745455	4.651131
31374	M Belgian-Style Barleywine	4.750000	4.857143	4.482143	4.803571	4.785714	28	4.735714	4.627120
3109	Armand'4 Oude Geuze Lente (Spring)	4.730769	4.730769	4.523077	4.669231	4.715385	65	4.673846	4.627069
39650	Pliny The Younger	4.600000	4.724590	4.482787	4.612295	4.723770	610	4.628689	4.623721
51855	Trappist Westvleteren 12	4.617925	4.718553	4.454009	4.633255	4.583333	1272	4.601415	4.599091

Logistic Regression Rating Prediction:

Logistic regression was done to the beer_review dataset in looking at how well we could predict overall beer rating by using features from our data set. Before I began building the logistic model I plotted a histogram of all of the review features (Appearance, aroma, taste, palate, overall) to look for collinearity of the features.

CDFs were completed for each feature with review_overall to see how each feature was related to the beers overall score. Review appearance and review overall had the closest similar distribution.



After plotting the CDFs I began building the logistic regression model. I created a dictionary for each feature and some combination of features to look at their predictive strength. The features consisted of: review_appearance, review_aroma, review_taste, review_palate, four_feature (a feature that consists of the previous 4 review characteristics), averaged (a feature consisting of the averaged total of the 4 review characteristics), beer_abv, and lastly combined which contained all of the features described. The accuracy of the predictions (prediction - actual) and Root Mean Squared Error (RMSE) are:

I then created binary categorical variables from the 103 total styles and ran another logistic regression using these and the previous combined features. The accuracy and RMSE for that model is:

```
encoded_accuracy
```

```
{'combined': 0.91018414445833373}
```

```
encoded_RMSE
```

```
{'combined': 0.29969293542168496}
```

This model results are based on an arbitrary ‘good’ target prediction that was set with an overall rating at or above 3.5. This value can be increased or decreased to change the accuracy (and RMSE) of the model. The results for each are as follows:

‘Good’ >= 4:

predictor_accuracy	RMSE
{'abv': 0.63041692490309564, 'appearance': 0.7031713184238999, 'aroma': 0.74000441190373645, 'averaged': 0.81014947950040439, 'base_features': 0.83207874197716314, 'combined': 0.83665031461075456, 'palate': 0.76866918075149426, 'taste': 0.82376335388720234}	{'abv': 0.6079334462726198, 'appearance': 0.54481986158371654, 'aroma': 0.50989762511337855, 'averaged': 0.43571839587007982, 'base_features': 0.40978196400383077, 'combined': 0.4041654183490288, 'palate': 0.48096862605424245, 'taste': 0.41980548604418888}

‘Good’ >= 4.5:

predictor_accuracy	RMSE
{'abv': 0.73708625271804784, 'appearance': 0.74740590564828724, 'aroma': 0.7603916089792746, 'averaged': 0.80682584535226953, 'base_features': 0.81813292435686003, 'combined': 0.82235364559807556, 'palate': 0.79137367774194567, 'taste': 0.80454005903547376}	{'abv': 0.51275115531995841, 'appearance': 0.50258739971443056, 'aroma': 0.4894981011410825, 'averaged': 0.43951581842719895, 'base_features': 0.42645876194907756, 'combined': 0.42148114358998839, 'palate': 0.45675630511034476, 'taste': 0.44210851718161481}

‘Good’ >= 5:

predictor_accuracy	RMSE
{'abv': 0.94255911425780226, 'appearance': 0.94227129006166166, 'aroma': 0.94225868462241458, 'averaged': 0.94705505425591141, 'base_features': 0.94695210983539402, 'combined': 0.94791642593779213, 'palate': 0.94250238978119061, 'taste': 0.94267046230448437}	{'abv': 0.23966828272050886, 'appearance': 0.24026799607592017, 'aroma': 0.24029422668384157, 'averaged': 0.23009768739404701, 'base_features': 0.2303212759703411, 'combined': 0.22821825970374904, 'palate': 0.23978659307561259, 'taste': 0.23943587386921705}

Content Recommendation Engine:

Content recommendation works through using product features to recommend similar products. For this project I used the beer_description data set. I used the description feature of this data set to build this recommendation system. An example of a beer description is,

Belgian Saison, “Beers in this category are gold to light amber in color. Often bottle-conditioned, with some yeast character and high carbonation. Belgian-style Saison may have Brettanomyces or lactic character, and fruity, horsey, goaty and/or leather-like aromas and flavors. Specialty ingredients, including spices, may contribute a unique and signature character. Commonly called “farmhouse ales” and originating as summertime beers in Belgium, these are not just warm-weather treats. US craft brewers brew them year-round and have taken to adding a variety of additional ingredients.”

Using these descriptions I created a Term Frequency-Inverse Document Frequency (TF-IDF) sparse matrix. This looks at each word, or n gram (n grams up to 3 were used in this instance), and creates a column with a count of each occurance. The rows represent individual documents or in this case beer description. Using this matrix of TF-IDF vectors I then computed the pair-wise similarity for each vector using a cosine similarity function. These pair-wise similarities are then compared and sorted by their similarity (most similar to least similar). These results are used to provide content ranked recommendations. For this project the result is 5 recommendation based on an individual style. Examples of this are:

```
get_recommendations('American Brown Ale')
9          American Black Ale
67                  Rye Beer
71     English Oatmeal Stout
100       Flanders Oud Bruin
6          English Brown Ale
```

```
get_recommendations('British Barleywine')
77           American Barleywine
70                American Stout
15    American Amber / Red Lager
86            English Strong Ale
1              German Doppelbock
```

After the content recommender was built I implemented a hybrid content-simple ranking recommendation engine for individual beers. This system took an input of a style, found the three most similar styles (not including input style) and output the top 3 beers from those styles ranked by their weighted average. Output examples for this hybrid system are:

```
beer_recommendation(q_ratings, style_recommendation('American Adjunct Lager'))
```

```
beer_name
Sam Adams Light             3.519
Elephant Beer               3.298
Baltika #9 Extra (Strong)   3.247
Sapporo Premium Beer        3.235
Asahi Super Dry             3.140
Amstel Light                2.928
Bud Light Lime              2.896
Pilsner                     2.478
```

Collaborative Filtering (by user ratings) Recommendations

This recommendation engine will look at how individual users rated beers and look for pair-wise similarities between the beers. Users that drank beer A and rated it well also drank beer B and rated it well. Users that drink beer A will be suggested beer B to try.

In the model that I built it does not look at the timing of ratings, such as a user trying B prior to A. Other limitations that were built into the matrix was limiting the users to having rated at least 100 beers and beers were required to have at least 10 ratings to be included. All ratings are all included as well. An issue that wasn't address were the beers that have identical names from different breweries. In future iterations of this engine these beers should be created with unique identifiers attaching beer_brewery or vice versa.

To build the matrix we drop all unnecessary columns from our beer_review data, these are 'brewery_id', 'brewery_name', 'review_time','beer_abv', 'beer_beerid', 'beer_style'. What we're left with is a data frame that looks like:

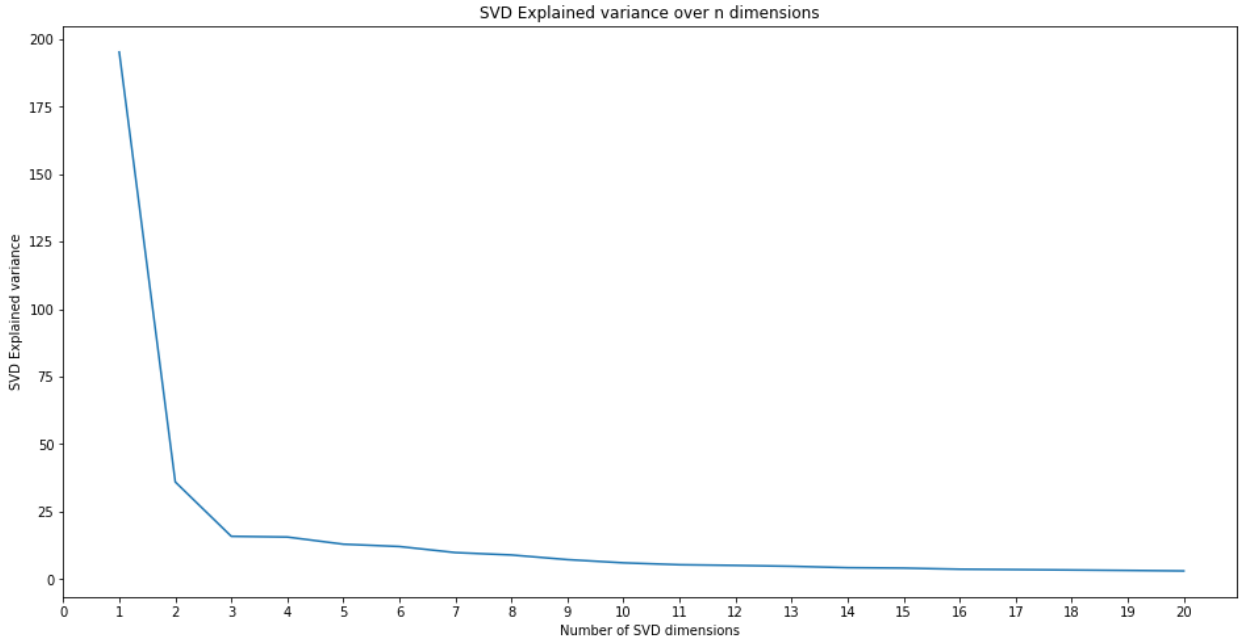
```
ratings.head()
```

	review_overall	review_aroma	review_appearance	review_profilename	review_palate	review_taste	beer_name
0	1.5	2.0	2.5	stcules	1.5	1.5	Sausa Weizen
1	3.0	2.5	3.0	stcules	3.0	3.0	Red Moon
2	3.0	2.5	3.0	stcules	3.0	3.0	Black Horse Black Beer
3	3.0	3.0	3.5	stcules	2.5	3.0	Sausa Pils
4	4.0	4.5	4.0	johnmichaelsen	4.0	4.5	Cauldron DIPA

From here I filtered out the UNKNOWN review_profilenames. I do this due to the sporadic nature of how beers may be rated under this umbrella moniker. After this is done the beers ratings undergo a combination, averaging, and weighting. They're filtered by users with over 100 users, and over 10 ratings per beer. At this point the data frame is pivoted and complete. Each column is an individual beer, and row a user (review_profilename). The data frame matrix looks like:

beer_name	"400" Ale	"Hop Obama" Ale	"Old Yeltsin" Imperial Stout	"Shabadoo" Black & Tan Ale	"The Wind Cried Mari..." Scottish Heather Ale	"True Blue" Blueberry Ale	# 100	#'s Ale	#9	*ellout*tout	...	Ølfabrikken Abbey Ale (Special Reserve)	Ølfabrikken Jule Ale	Ølfabrikken Kloster Jul
review_profilename														
0110x011	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000
05Harley	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000
100floods	0.000	0.000	0.000	0.000	0.000	0.000	4.300	0.000	0.000	0.000	...	0.000	0.000	0.000
1759Girl	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000
1Adam12	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	...	0.000	0.000	0.000

Once the matrix was built I went on to creating the recommendation engine. First we transposed our User-Item matrix to Item-User. Similar to when I built the TF-IDF matrix, our ‘target’, specific beers, will be the rows to the matrix. To reduce model complexity I fit and transformed the matrix using Single Value Decomposition (SVD).



n_components was originally set at 3 the higher range of where the ‘elbow’ is found. After this is completed the matrix is ran through a Pearson product-moment correlation coefficients. This will provide a matrix of pair-wise similarity computations that we can use to create recommendations with. Mapping the indices recovers the original beer labels. The correlation coefficients are then sorted and presented as the recommendations for any individual beer:

```
recommendations('Furious')
```

```
['Bender',
 'Samuel Adams Utopias',
 'Coffee Bender',
 'CynicAle',
 'Anniversary Ale',
 'Darkness',
 'Kona Coffee Macadamia Coconut Porter']
```

This sort of collaborative filtering can suffer from the ‘sequel’ or grouping problem. Where items (beers) that are in a series or grouping will be paired together. This can be seen in the Furious recommendation where Bender, Coffee Bender, CynicAle, and Darkness are from the same brewery, Surly Brewing.

Evaluations and Similarity Metric trials

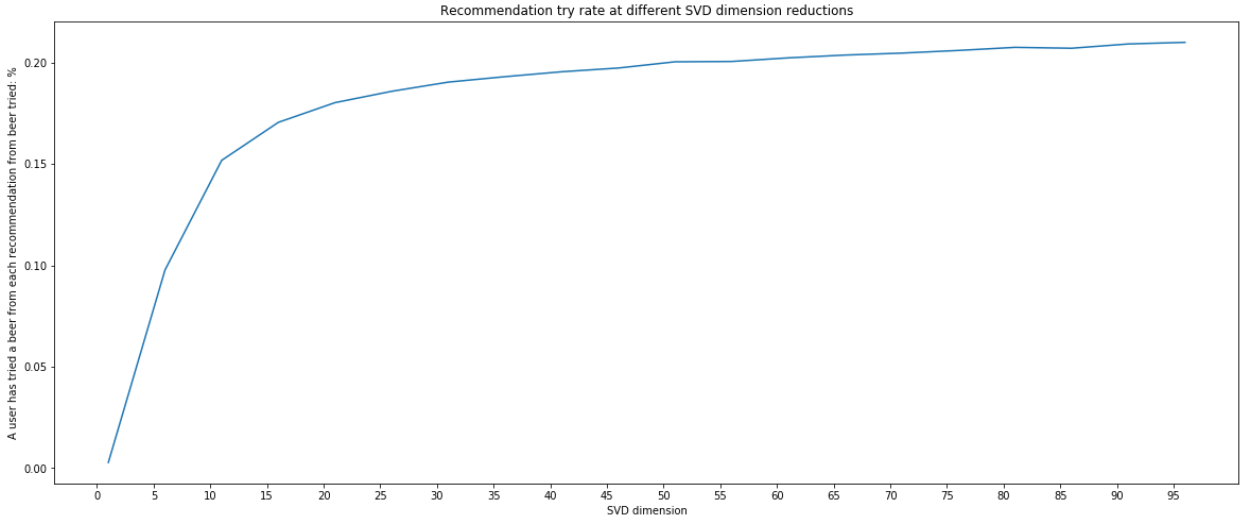
The model was built using users with over 100 ratings, all testing was done with users in a rating range of 2-99.

Two evaluation metric are used. The primary metric is Mean Average Precision at cutoff k (MAP at k). This metric takes the list of beers rated (N), it creates 7 (k) recommendations per beer in N (N_1, N_2, … N_i). This results in (kN_1, kN_2, … kN_i). It then checks to see if any recommendations in the set kN_i exist within N. The benchmark is increased by 1 for every recommendation set that exists within N. The benchmark is then divided by N. This precision is summed together for each user and finally divided by total users in the data set.

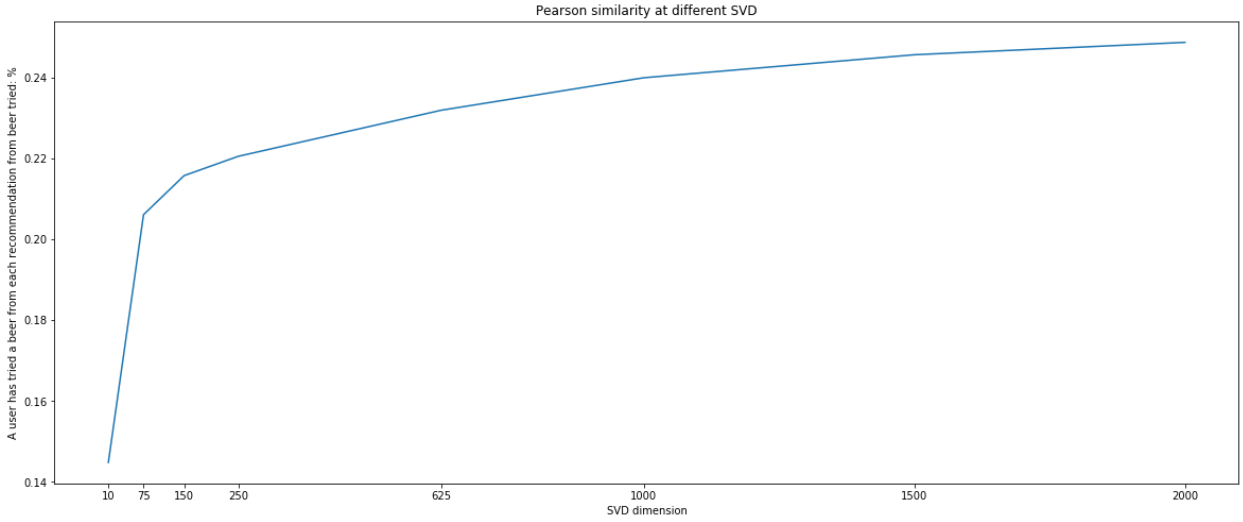
Secondly a 'has a user tried' any recommended beers from total list of rated beers. This metric take all beers rated (N) makes 7 recommendations for each beer rated and checks the list of N if any of the recommendations are within the list of N.

SVD evaluation

Early indications showed that a SVD n_components may be ideal at 3 dimensions. Later testing showed issues with this. While still using Pearson Correlation similarity I tested n_dimensions from 1 to 95 with the following results:



This shows that there was still significant increases in Mean Average Precision at cutoff k (MAP at k). The increases begin to slow around 10 dimensions and significantly tail off around 20-25 dimensions. We can see that this continues to be the case even for large dimensions (2000+)



Evaluating different similarity measures in regard to SVD
The three similarity measures that were looked at are Pearson Coefficient, Cosine, and Euclidean.
To review how MAP at k peaks in regard to SVD with Pearsons

SVD:

	`[rec for rec in recommendation_precisio`	`[tries for tries in user_has_tried]`
10	`[0.1448259169998671,`	`[0.4327488343807014,`
75	`0.20608076717461243,`	`0.5054226636934928,`
150	`0.21574358032973792,`	`0.5181937968781675,`
250	`0.22051583675231115,`	`0.5233630650719643,`
625	`0.23190753949614912,`	`0.5374518548550578,`
1500	`0.2399491201303128,`	`0.5461686600445976,`
2000	`0.2456376271487097,`	`0.5521994729373606,`
	`0.24870032390744395]`	`0.5558990472329212]`

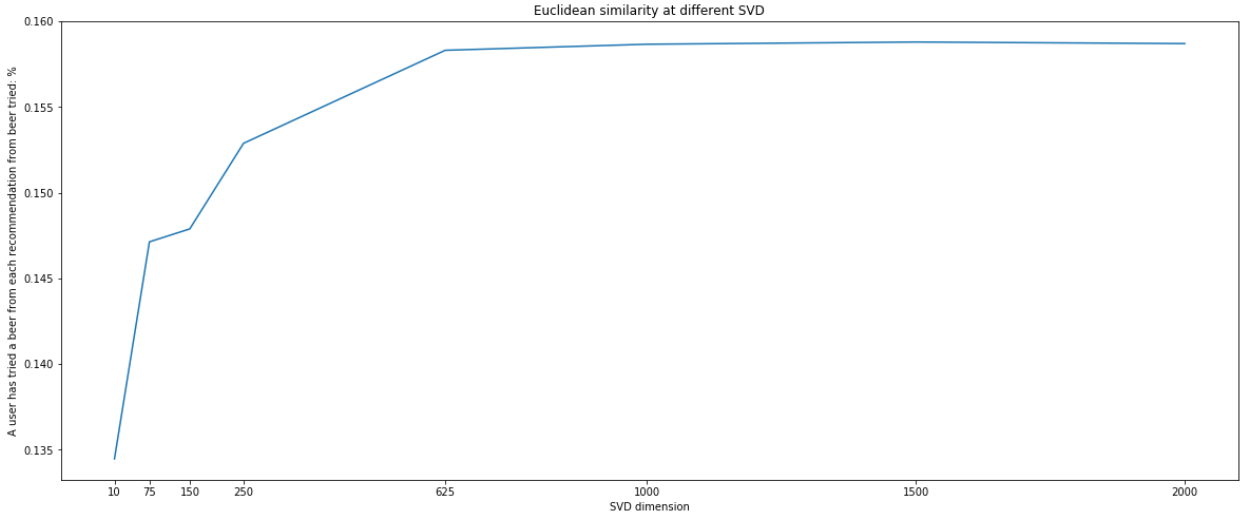
MAP at k with Cosine

SVD:

	[rec for rec in recommendation_precision]	[tries for tries in user_has_tried]
10	[0.2784916558856606,	[0.4554530711534563,
75	0.29719417898303024,	0.5200182444759781,
150	0.30448593531735446,	0.5622846138252585,
250	0.3333283314228271,	0.63744171903507,
625	0.35047496954131924,	0.6777822825866613,
1500	0.3663965116665127,	0.7009933103588081,
2000	0.38714345232986447,	0.7268396513277924,
	0.3893567541394226]	0.727802554226637]

We can see with cosine a fairly significant increase of .0538 (5.38%) in MAP at k from 250 to 1500 dimensions. Cosine also shows the most accurate model topping at around 38% MAP at k and in 73% cases a user as rated a beer and rated another beer that was in the set of recommended beers. Even with minimum dimensions of 10 we see that cosine similarity out performs the top performance of both Euclidean (16%) and Pearson (25%).

MAP at k with Euclidean



SVD:

```
[rec for rec in recommendation_precision]
```

10
75
150
250
625
1500
2000

```
[0.13447357808081242,
 0.14713357169404492,
 0.14788712745881658,
 0.15288119666113095,
 0.15830741520537212,
 0.1586606432334881,
 0.15879016985574654,
 0.15869773507285637]
```

With Euclidean similarity we see increases in MAP at k slow and then pick up again until 625 dimensions. The empirical difference we see from 75 dimensions to 625 is only .0115. Depending how how accurate someone was looking to get the model compared to computing costs will dictate which dimension would be best. In Euclidean similarity we also see the only decrease in MAP at k with increasing dimensions between 1500 and 2000.

Summary and future additions

During this project I built 4 different recommendation systems, simple, content, hybrid, collaborative filtering. Each system has pros and cons, the content recommender is stand alone and doesn't require and user input. Collaborative filtering is complex but has a larger breadth of recommendation power and evaluation metrics. Simple is easiest to implement but the most limited when it comes to actual recommendations. The collaborative filtering was found to work best using cosine similarity at around 1500 dimensions which is computationally expensive; however, even at 10 dimensions at the cost of 11% accuracy we still outperformed the other similarity metrics (Pearson, Euclidean).

Future implementations will include Manhattan distance and Jaccard similarity measures. Better evaluation metrics would allow for increased parameter tuning. Beyond these evaluation improvements is building a hybrid collaborative filtering that takes into account the logistics regression predictive ability. This would open up the possibilities to create recommendations, create predictive rankings for them and evaluate them to how users actually rated them in either hold out test sets or a leave out out strategy. This could fine tune recommendations on beer relevance. The current model only addresses similarity to user rating patterns, not on user ratings themselves.

The age of the data is something that in future analysis can and should be looked at. In the United States different legal barriers were removed and market entry costs were decreased. This caused a massive uptick in craft brewing that the current data doesn't have.

Other details that would be powerful is regionality. Knowing where beers are available geographically (i.e. internationally, nationally, regionally, brewery specific) would allow the recommendation engine to take that into account. Users would not be recommended beers that they don't have access to. This would require knowing where users are located geographically too. Additional incentive to add this to the model would be the accuracy increase that could be seen by the recommendation model. Since beers that users don't have easy access to are removed there is increased likelihood that the recommended beers may be tried due to accessibility. This creates a more powerful model and drives user reward for using the model.

Another filtering procedure that should be done is on availability of the beer in its market. This is referring to whether the beers are limited releases or retired. Many of these kind of beers are highly rated but impossible (highly improbable) for future users to encounter them.