Beer Recommender System

Including analysis of over 1.5 million beer reviews

Data Acquisition and Cleaning

This project uses over 1.5 million beer reviews from BeerAdvocate, hosted on Kaggle.

I was inspired to use this dataset after reading a great article by Tanya Cashorali on how to interview for data science hires:

https://www.linkedin.com/pulse/how-hire-test-data-skills-one-size-fits-all-interview-tanya-cashorali/

Dataset Features

1. Brewery ID - an integer identifier for each unique brewery
2. Brewery Name - a string containing the name of the brewery
3. Review Time - an integer containing the date the review was submitted
4. Review Overall - a float of the complete review score. Ratings in this dataset are from 1-5
5. Review Aroma - a float of the review score for the aroma of the beer
6. Review Palate - a float of the review score for the mouthfeel of the beer
7. Review Appearance - a float of the review score for the appearance of the beer
8. Review Taste - a float of the review score for the taste of the beer
9. Review Profilename - a string containing the reviewer's username
10. Beer Name - a string of the reviewed beer's name
11. Beer ID - an integer identifier for each unique beer
12. Beer Style - a string containing the style of the beer
13. Beer ABV - a float denoting the reviewed beers percent alcohol by volume

Missing Values

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1586614 entries, 0 to 1586613
Data columns (total 13 columns):
brewery_id            1586614 non-null int64
brewery_name          1586599 non-null object
review_time           1586614 non-null datetime64[ns]
review_overall        1586614 non-null float64
review_aroma          1586614 non-null float64
review_appearance     1586614 non-null float64
review_profilename    1586266 non-null object
beer_style            1586614 non-null object
review_palate         1586614 non-null float64
review_taste          1586614 non-null float64
beer_name             1586614 non-null object
beer_abv              1518829 non-null float64
beer_beerid           1586614 non-null int64
dtypes: datetime64[ns](1), float64(6), int64(2), object(4)
memory usage: 157.4+ MB
```

Accounting for missing values

The Dataset contains around 68,000 missing values, mostly from the beer_abv column. Since there is already so much data to work with, I decided to remove these altogether, as well as any reviews scoring less than 1.

There were also duplicate reviews present. I removed all duplicates, keeping the most recent review.

Distribution of Reviews

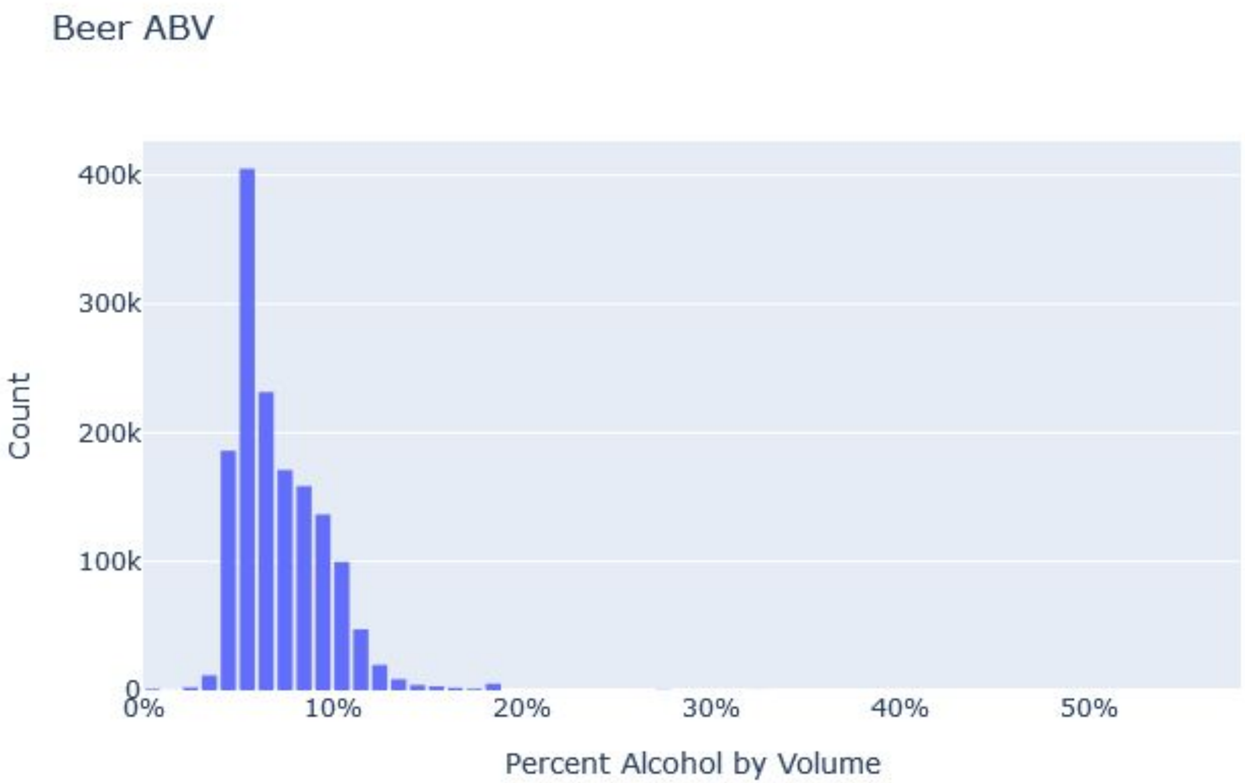
Distribution of Reviews - Highly Rated

Distribution of Reviews - Low Ratings

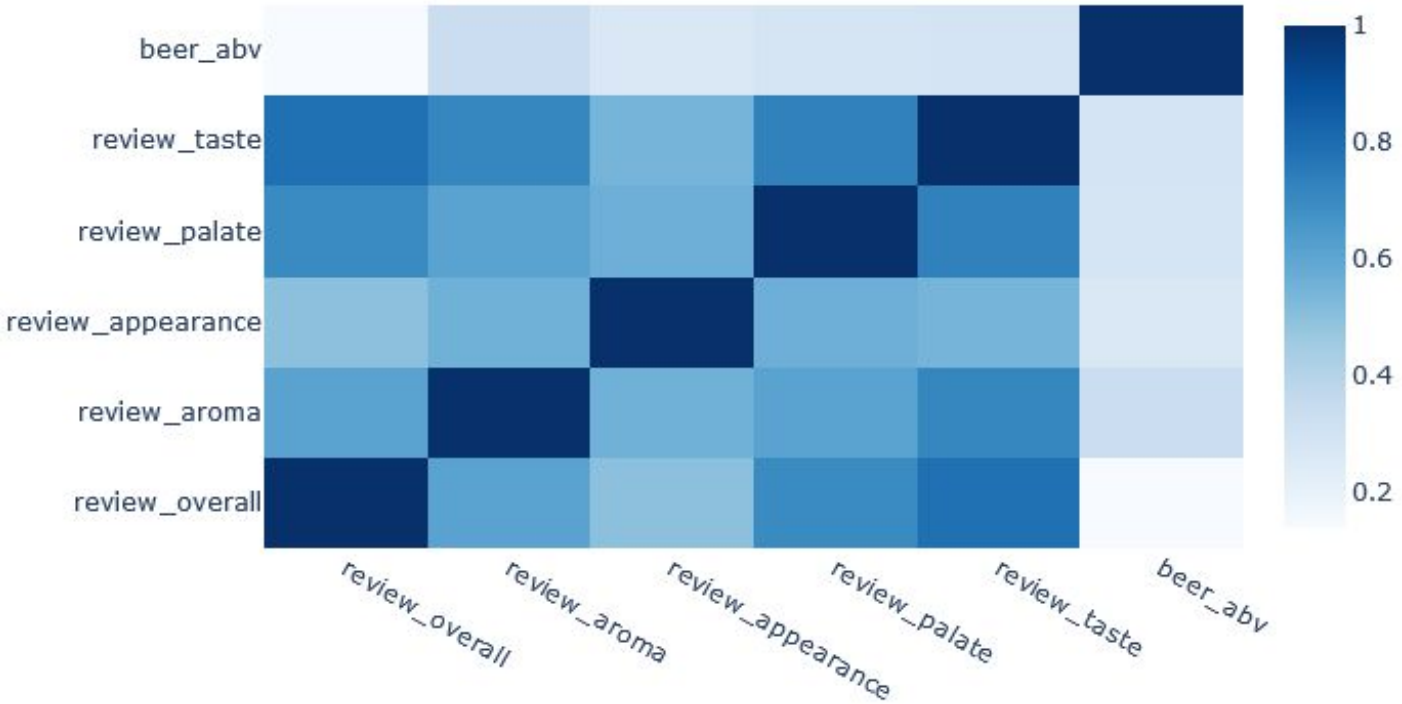
Reviews Analysis

- Overall, there are more positive than negative reviews
- Review taste appears to be most closely related to review overall
- Review appearance is mostly independent of the overall review score.

Alcohol By Volume Distribution



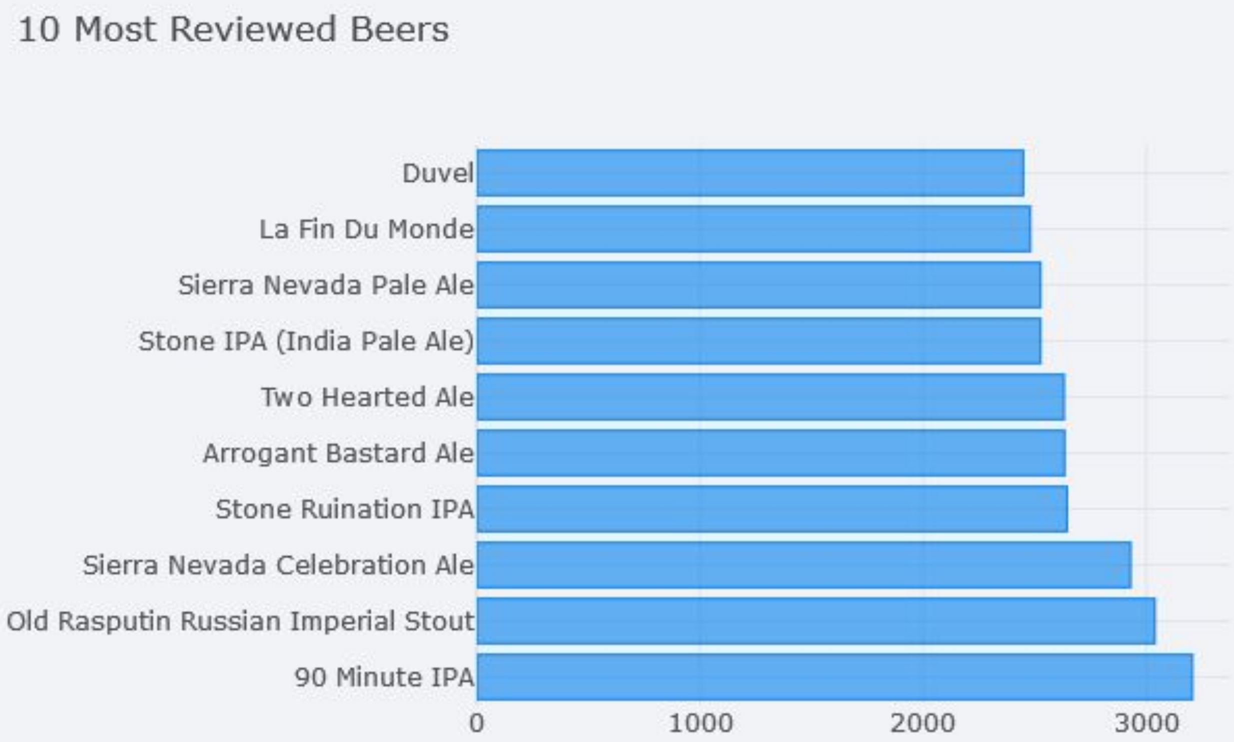
Correlation Heatmap



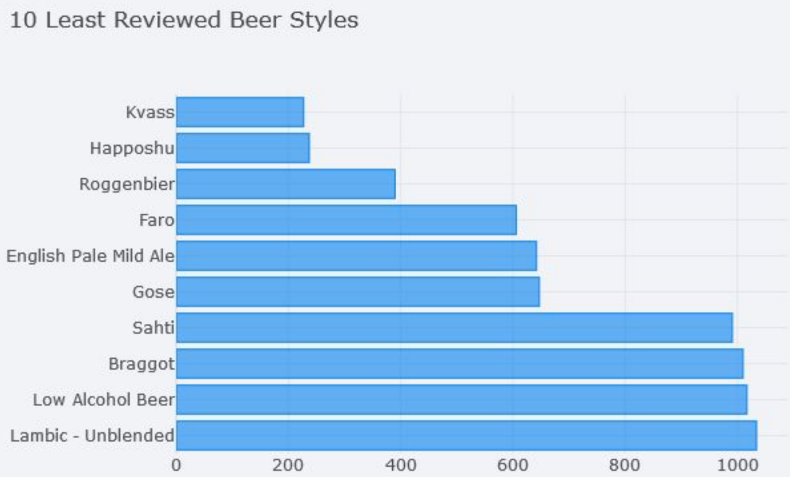
Correlation Analysis

- Review taste has the strongest correlation with review overall
- Review palate has the second strongest correlation
- Review appearance and aroma have weak correlation
- Beer ABV has no correlation

Most Reviewed Beers

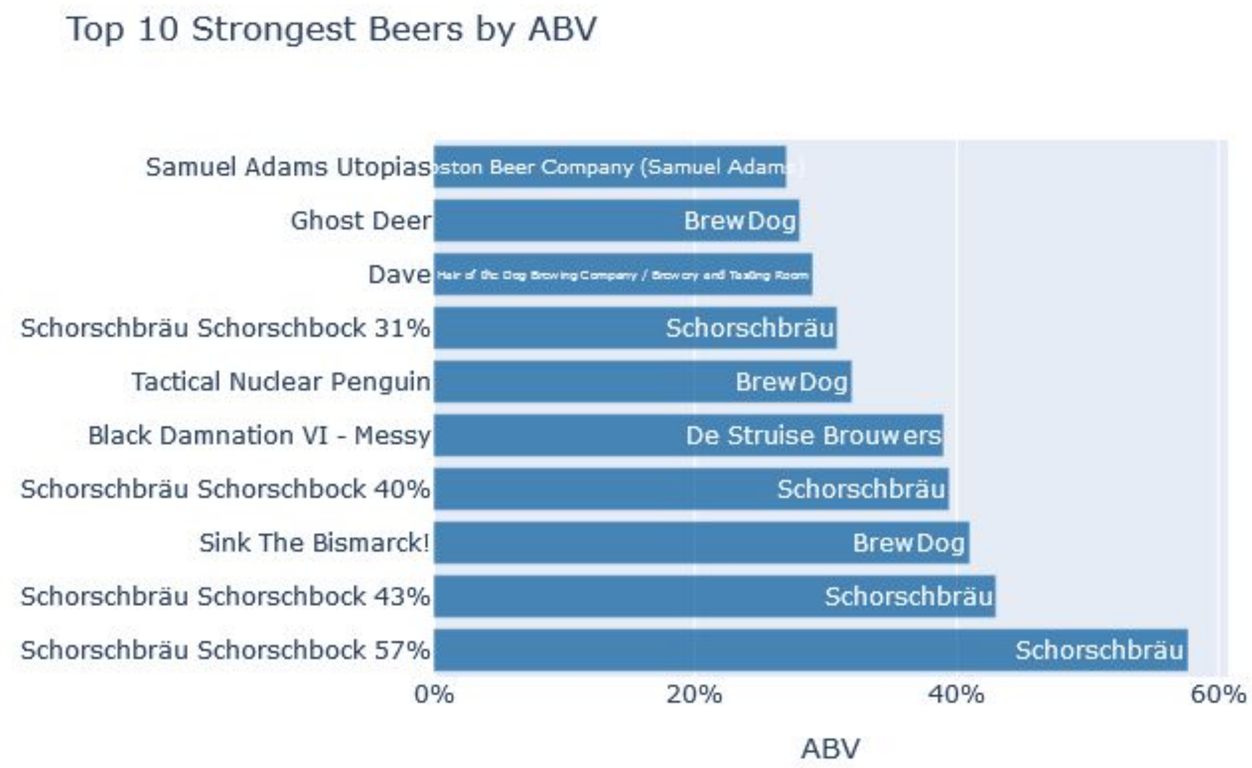


Beer Style Review Counts



Most Reviewed Breweries

Strongest Beers By ABV



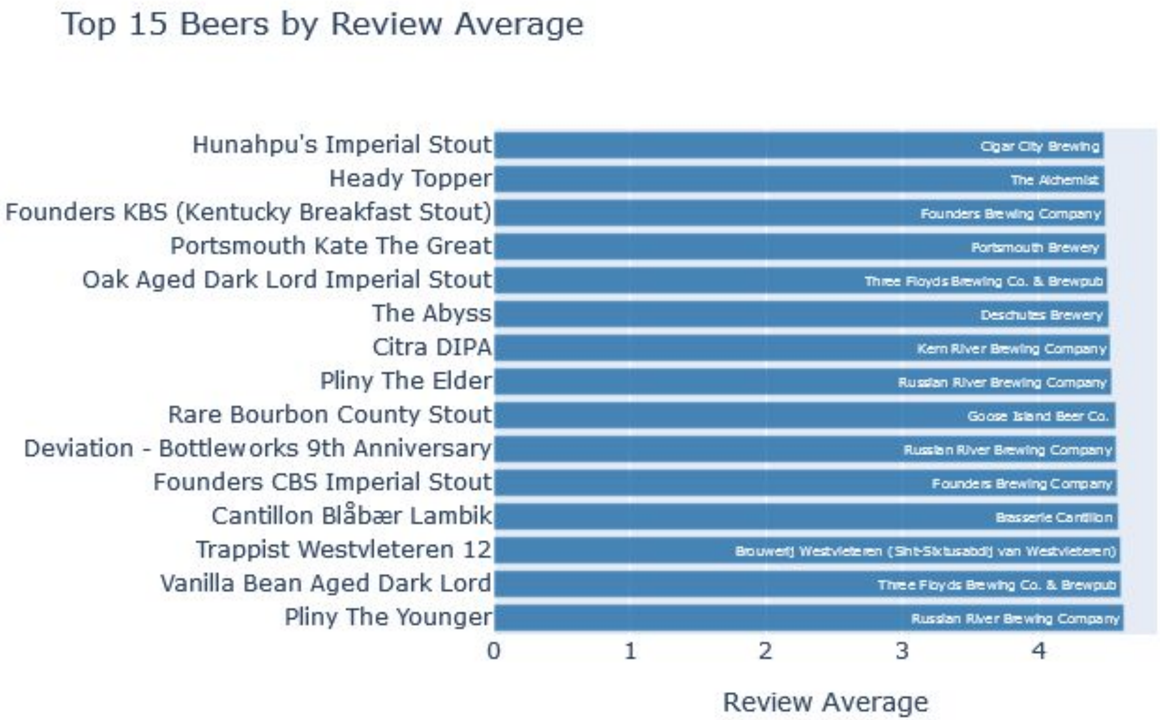
Establishing a ranking system

When sorting by mean review overall, the results are thrown off by beers with a low number of ratings.

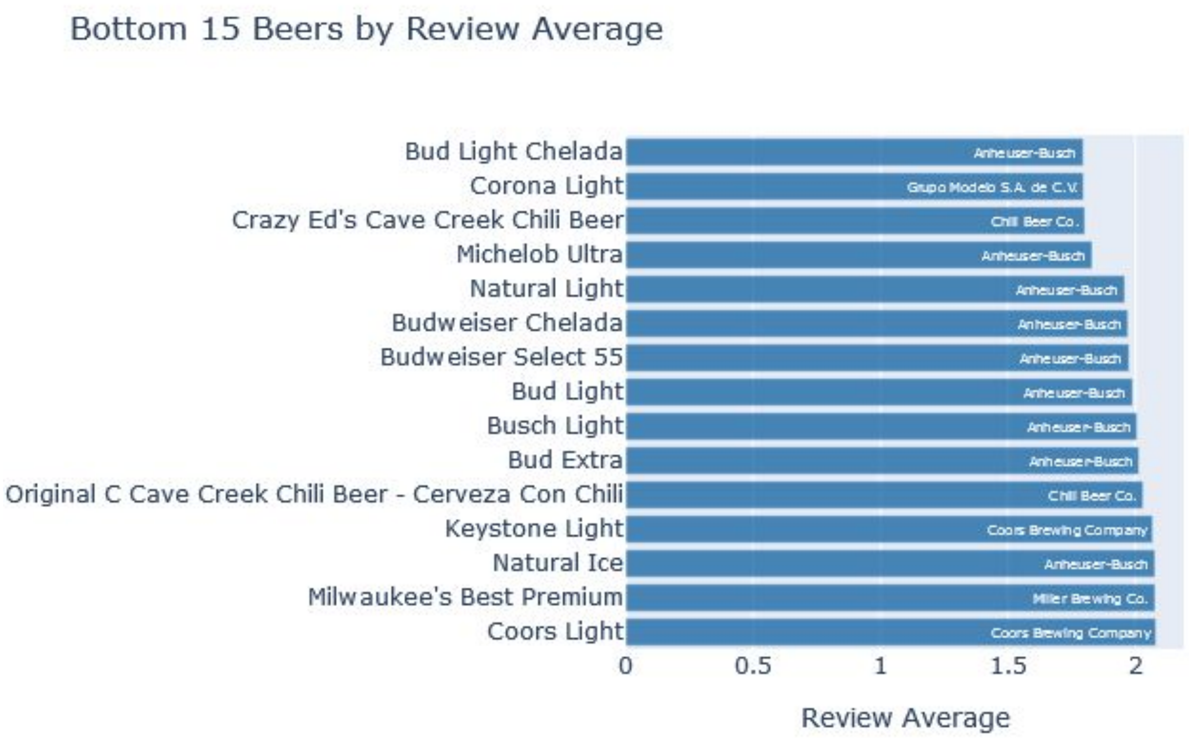
To address this, I created the review_average column, which consists of the average of all review categories, and the review_total column.

I then removed all beers with less than 100 reviews and ordered by mean review_average

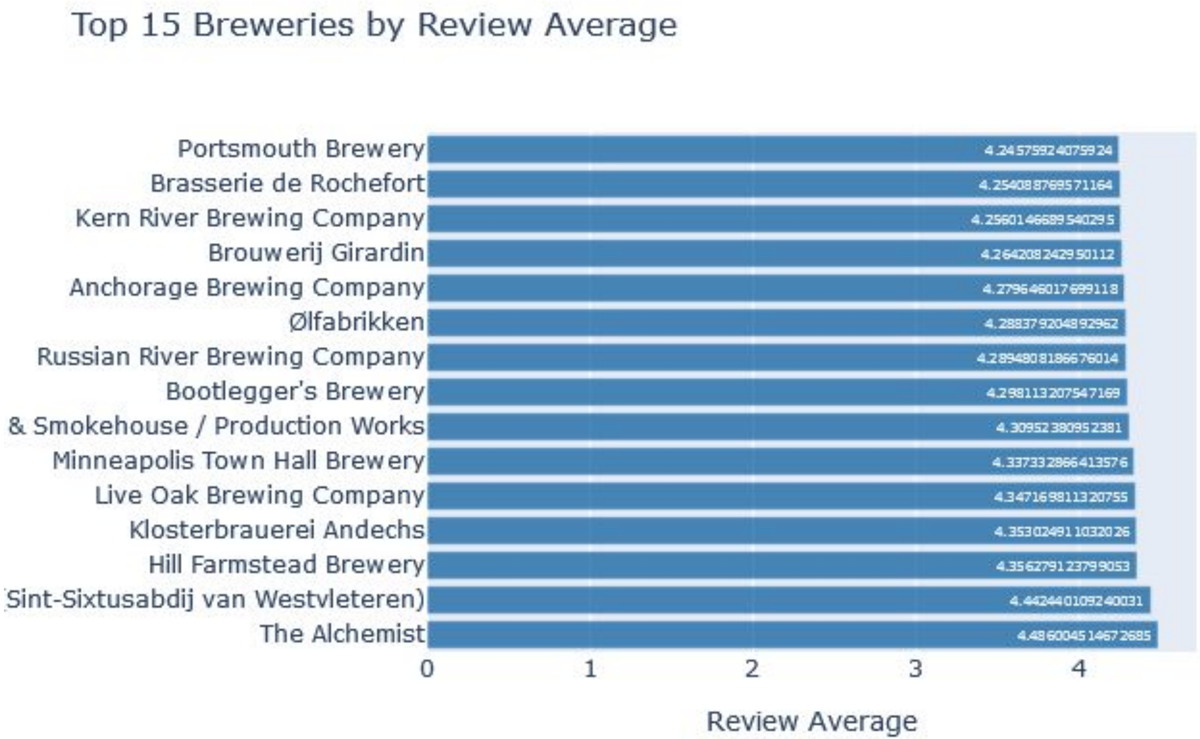
Top Beers by Review Average



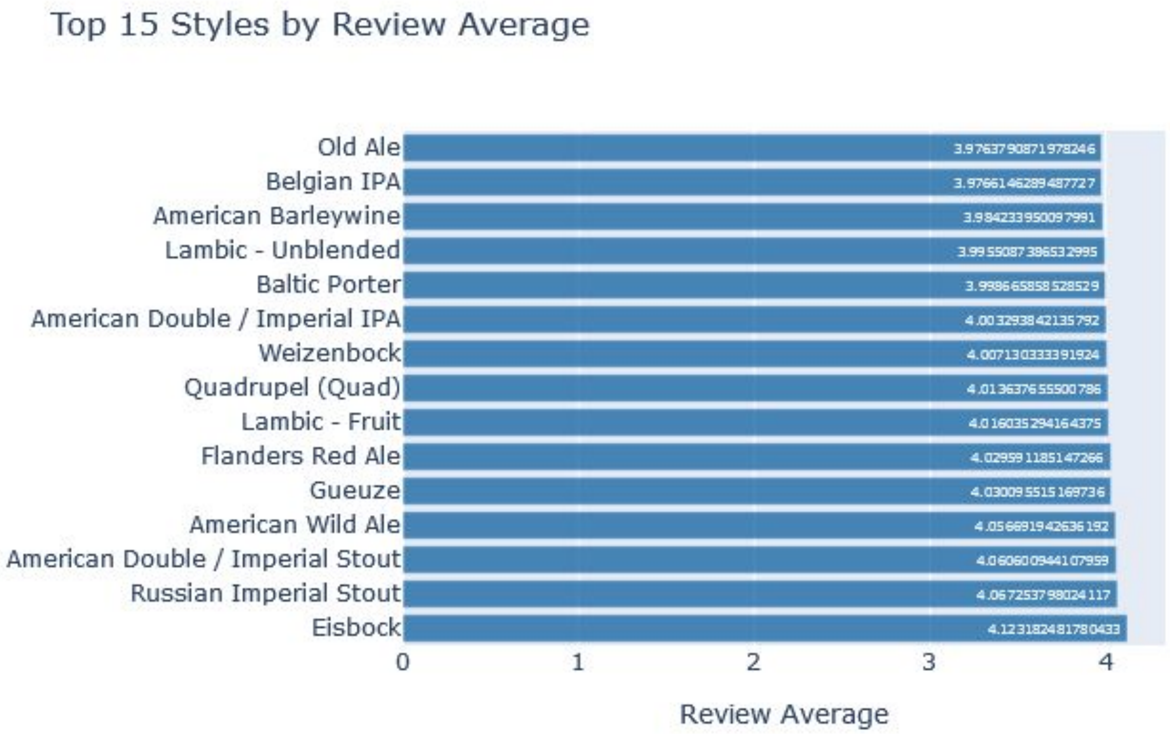
Bottom Beers by Review Average



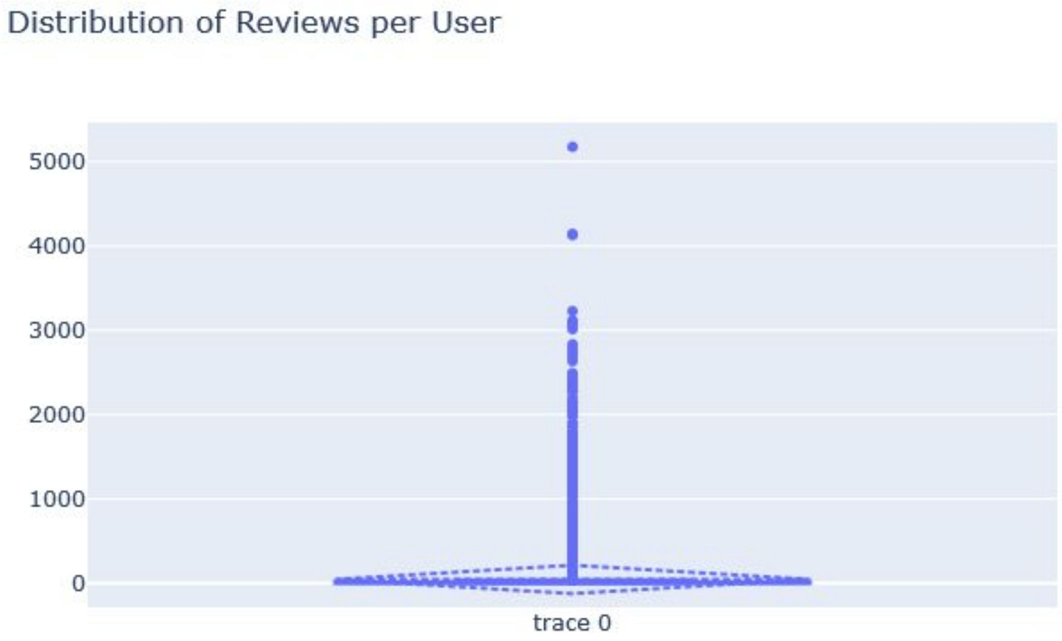
Top Breweries by Review Average



Top Styles by Review Average



Reviews Per User

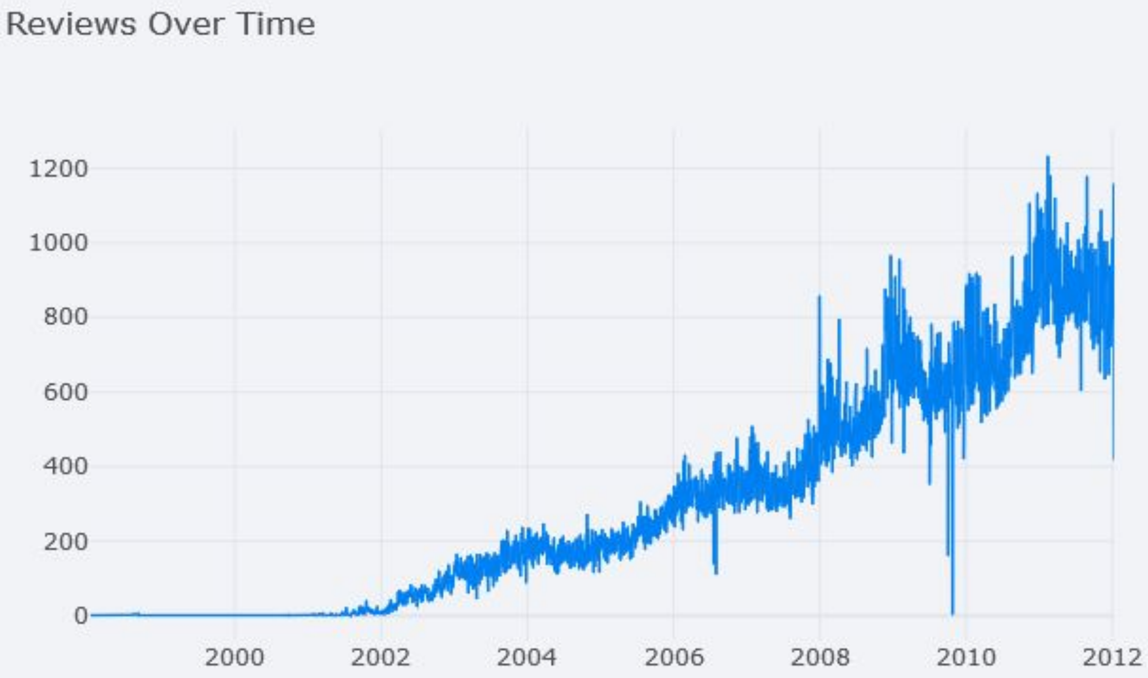


User Reviews

The mean amount of reviews per user is 45, but the median is only 3

This means that there are a small subset of highly prolific users with many more reviews than average.

Reviews Over Time



Recommendation System

To create the recommender system, I created a new dataframe containing only the relevant columns, and removed all beers reviewed less than 50 times.

	review_profilename	beer_name	review_overall
0	stcules	Sausa Weizen	1.5
1	stcules	Red Moon	3.0
2	stcules	Black Horse Black Beer	3.0
3	stcules	Sausa Pils	3.0
4	johnmichaelsen	Cauldron DIPA	4.0

Initial Fit

An initial fit with an untuned SVD algorithm achieved RMSE of 0.575. Here's a look at the first predictions it made:

```
[Prediction(uid='Long813', iid='Hophead Double India Pale Ale', r_ui=4.0, est=3.585667317417661, details={'was_impo
ssible': False}),
 Prediction(uid='beerwolf77', iid='Orchard White', r_ui=4.0, est=4.007106415545197, details={'was_impossible': Fals
e}),
 Prediction(uid='SShelly', iid='Green Flash Le Freak', r_ui=4.0, est=4.131120724067257, details={'was_impossible':
False}),
 Prediction(uid='maxpower', iid='Lump Of Coal', r_ui=3.5, est=3.539782794389326, details={'was_impossible': Fals
e}),
 Prediction(uid='Cs1987', iid='König Pilsener', r_ui=4.0, est=3.6295640690908293, details={'was_impossible': Fals
e})]
```

Testing Different Algorithms

Algorithm	test_rmse	fit_time	test_time
SVDpp	0.569573	3487.744543	121.888504
BaselineOnly	0.573327	2.806333	4.093334
KNNBaseline	0.573759	12.133106	114.039659
SlopeOne	0.574417	18.701889	114.540148
SVD	0.575154	38.108621	3.073264
NMF	0.584200	42.037437	3.236865
KNNWithMeans	0.586201	9.753362	105.637112
KNNBasic	0.588927	9.580004	100.474461
KNNWithZScore	0.589756	10.216587	109.887675
CoClustering	0.625232	14.011497	3.729301
NormalPredictor	0.953737	1.216606	3.308804

SVD++ Grid Search

I attempted a grid search to tune the SVD++ hyperparameters

```
param_grid = {'n_factors': [50,100,150],
              'n_epochs': [20,30],
              'lr_all': [0.005,0.01],
              'reg_all':[0.02,0.1]}
```

However, this grid search was incredibly computation intensive and after several days, I decided to perform manual tuning instead.

SVD++ Parameters and Performance

```
trainset, testset = train_test_split(data, test_size=0.25)
algo = SVDpp(n_factors=150, n_epochs=75, lr_all=0.01, reg_all=0.1)
predictions = algo.fit(trainset).test(testset)
accuracy.rmse(predictions)
```

```
RMSE: 0.5718

0.5718358448010138
```

```
algo = NormalPredictor()
predictionsnormal = algo.fit(trainset).test(testset)
accuracy.rmse(predictionsnormal)
```

```
RMSE: 0.9570

0.9569947792018437
```

SVD++ Parameters and Performance

This model achieved an RMSE of 0.571, outperforming the NormalPredictor by around 0.4%.

Best Predictions

	review_profilename	beer_name	rui	est	details	items_rated	num_ratings	error
139103	oteyj	The Abyss	5.0	5.000000	{'was_impossible': False}	55	795	0.000000
260452	oteyj	Cantillon Crianza Helena	5.0	5.000000	{'was_impossible': False}	55	48	0.000000
211923	oteyj	Supplication	5.0	5.000000	{'was_impossible': False}	55	654	0.000000
204248	oteyj	Trappist Westvleteren 8	5.0	5.000000	{'was_impossible': False}	55	440	0.000000
262703	whartontallboy	Uerige Altbier (Classic)	4.0	3.999997	{'was_impossible': False}	241	129	0.000003
140207	mikesgroove	The Reverend	4.0	4.000004	{'was_impossible': False}	2227	448	0.000004
205873	brewandbbq	Corne De Brume	4.0	4.000006	{'was_impossible': False}	507	39	0.000006
181089	brentk56	J.W. Lees Harvest Ale (Port Cask)	4.0	4.000007	{'was_impossible': False}	1800	106	0.000007
144698	Foxman	Allagash Fluxus 2007	4.0	4.000008	{'was_impossible': False}	541	40	0.000008
178598	kbub6f	Prohibition Ale	4.0	4.000009	{'was_impossible': False}	395	184	0.000009

Worst Predictions

	review_profilename	beer_name	rui	est	details	items_rated	num_ratings	error
188593	aaronh	Drie Fonteinen Oude Geuze	1.0	4.197010	{'was_impossible': False}	406	323	3.197010
138373	dasenebler	YuleSmith (Summer)	1.0	4.241274	{'was_impossible': False}	352	569	3.241274
88273	rvdoorn	Darkness	1.0	4.274347	{'was_impossible': False}	197	391	3.274347
14188	rye726	Uerige Doppelsticke	1.0	4.277991	{'was_impossible': False}	732	244	3.277991
228632	EssexAleMan	Hardcore IPA (2nd Ed. 9.2%)	1.0	4.353312	{'was_impossible': False}	62	64	3.353312
89760	ChrisCage	La Fin Du Monde	1.0	4.356200	{'was_impossible': False}	123	1438	3.356200
32019	brdc	Sinners Blend 2008	1.0	4.374767	{'was_impossible': False}	611	39	3.374767
206510	jfitzy78	Fantôme Brise-BonBons	1.0	4.408655	{'was_impossible': False}	35	105	3.408655
32728	rvdoorn	Pliny The Elder	1.0	4.479191	{'was_impossible': False}	197	1248	3.479191
190299	madtappers	The Dissident	1.0	4.522709	{'was_impossible': False}	33	231	3.522709

Obtaining the Top Predictions Per User

```
from collections import defaultdict

def get_top_n(predictions, n=10):

    # map preictions to each user.
    top_n = defaultdict(list)
    for review_profilename, beer_name, true_r, est, _ in predictions:
        top_n[review_profilename].append((beer_name, est))

    # sort predictions for each user and retrieve the k highest ones.
    for review_profilename, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[review_profilename] = user_ratings[:n]

    return top_n
```

Top Predictions Per User

```
top_ratings['whartontallboy']
```

```
[('St. Bernardus Abt 12', 4.290482442303683),
 ('Alpha King Pale Ale', 4.2616011463916),
 ('Gumballhead', 4.2563432290147981),
 ('Founders Breakfast Stout', 4.240451085670301),
 ('Southampton Saison', 4.238378760158777),
 ('Péché Mortel (Imperial Stout Au Cafe)', 4.18132717070748399),
 ('YuleSmith (Summer)', 4.180392408293553),
 ("Samuel Smith's Oatmeal Stout", 4.158039115939 4585),
 ('Southampton Grand Cru', 4.154582216869394),
 ('Tripel Karmeliet', 4.145165485014519)]
```