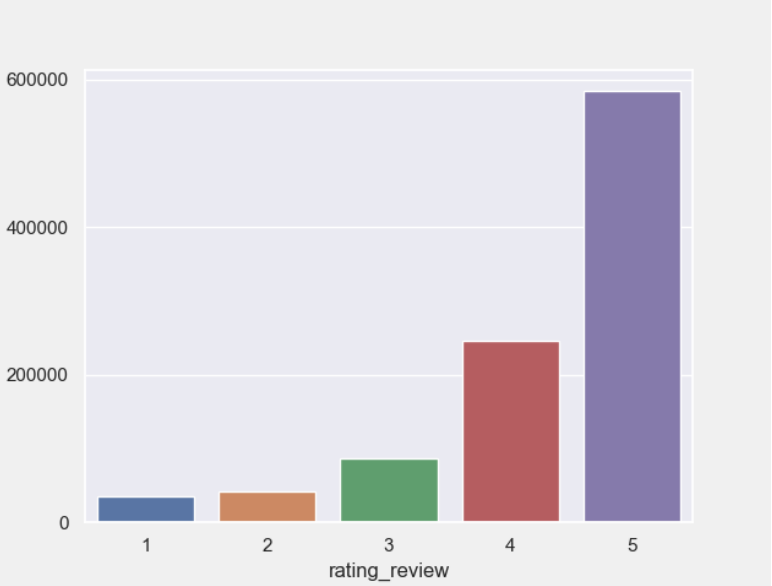
EDA:

Some simple steps were taken early on to remove bad data from the recommendations.

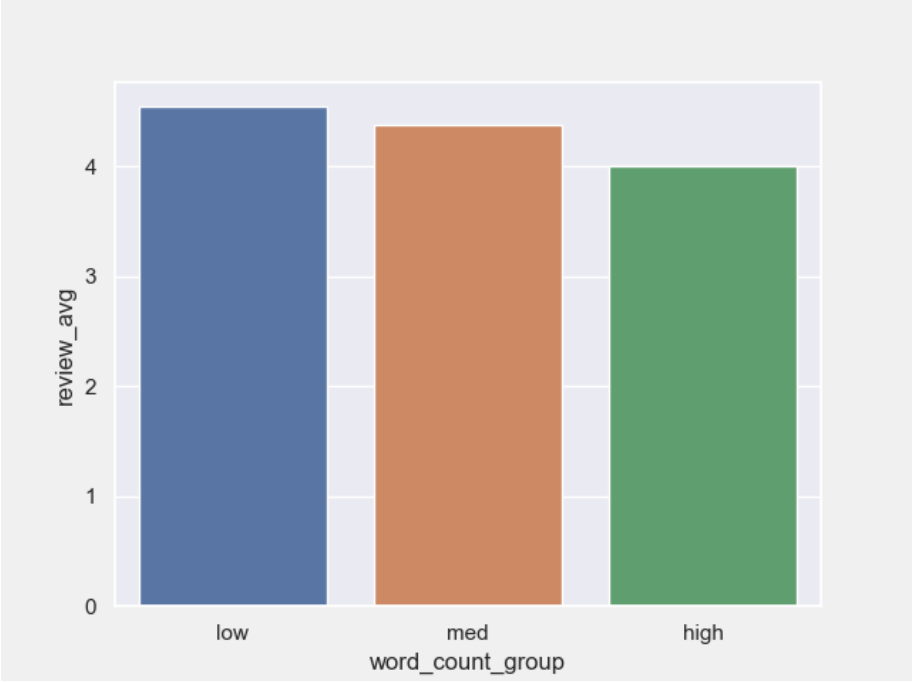
Any observations with blank reviews or scores were excluded from the analysis as there is no reliable way to infer what they would have said.



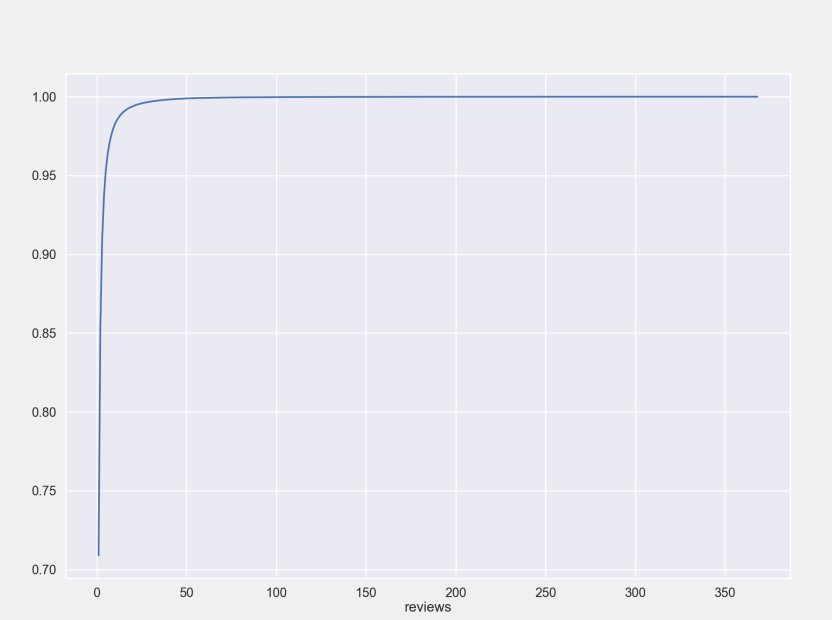
To start with some simple questions were asked of the data. The simplest being what is the distribution of our target variable.

This shows that people tend to rate restaurants well. Really calling into question the thought process of people only post a review when they want to complain about something.

Looking at the reviews this time I wanted to know what the average length of the reviews and how does the review score average change as you increased the number of words in the review.

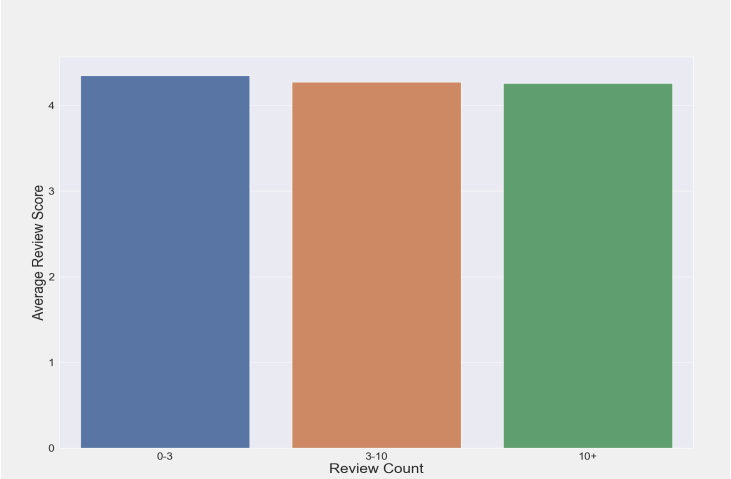


As demonstrated by the figure above the longer a review is the more likely it is to be negative. So while people rate more things highly they have a lot more to say about bad reviews.

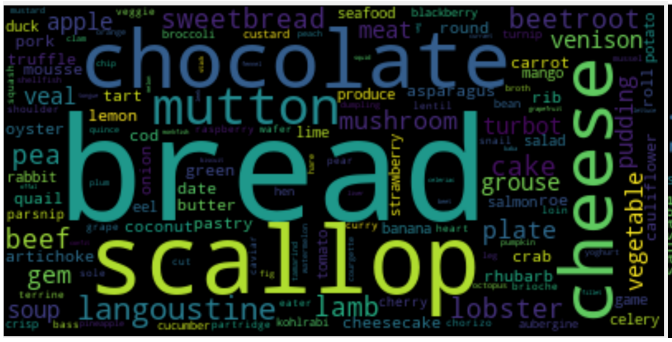


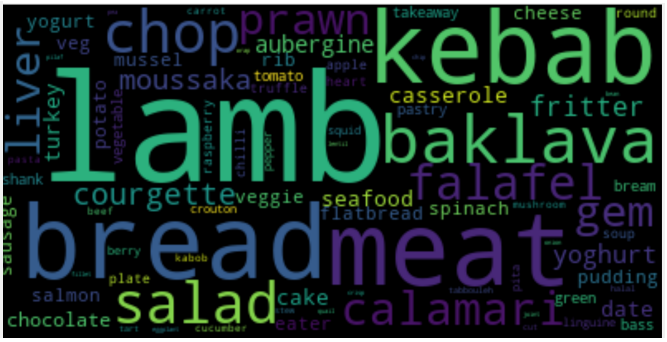
Next I took a look at the Reviewers themselves. And found out that a vast majority of users only have less than 10 review

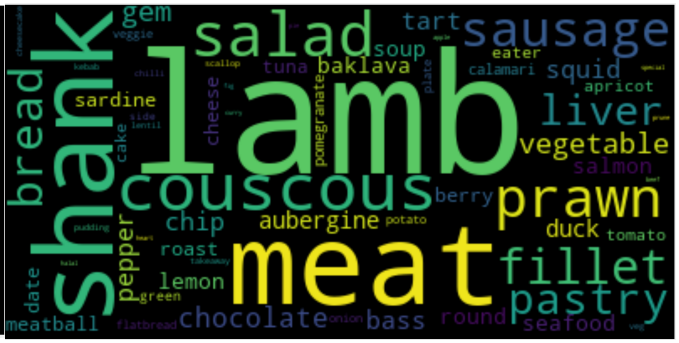
And looking at the scores users give. The more reviews given seems to show a trend to more honest or at least lower rated reviews. Even if the trend is very small.

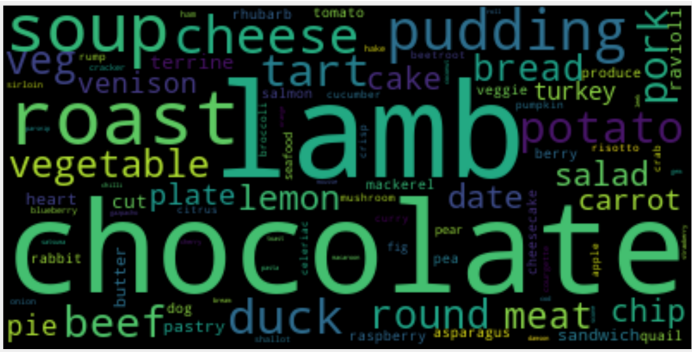


As for the restaurants, some feature analysis of the top and bottom restaurants with more than 500 reviews shows us that there is very little domination of restaurant brand in London. As you can see for the word clouds below the Top 5 cover a wide range of food and cuisine.

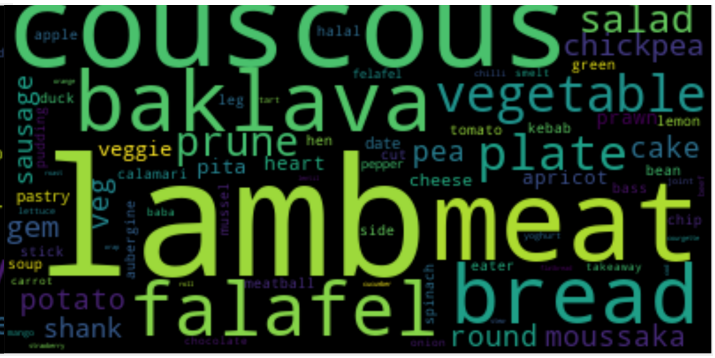








Just like the top rated restaurants there is no stand out worst cuisine in london either as they all offer different things, suggesting that in London you can always find somewhere good to eat in whatever style you want if you are willing to look.



Model:

I decided to use a grid search approach to a Random Forest Classifier as they a known to be quite good at multi-type classification problems, where as if we were attempting just positive or negative sentiment analysis then something like a Logistic Regression or Nieve Bayes approach would have likely been appropriate.

Calibrated Model

The pickle file that was sent with this report and github link is for the api. I understand you want to promote code and not models to a production environment just so that any idiosyncrasies between dev / stage / prod don't cause strange drift in the model. But just incase there is a bug / runtime error in my code that I missed that means you can't calibrate a model using it yourselves:

Confusion Matrix	**1**	**2**	**3**	**4**	**5**
1	3452	770	1160	532	1317
2	1427	1161	2835	1258	1674
3	596	687	5758	5545	4744
4	147	115	1827	15983	31250
5	97	39	502	8041	108393

Reveiw Score	**precision**	**recall**	**f1-score**	**support**
1	0.60	0.48	0.53	7231
2	0.42	0.14	0.21	8355
3	0.48	0.33	0.39	17330
4	0.51	0.32	0.40	49322
5	0.74	0.93	0.82	117072
Accuracy Total			0.68	199310

API:

Deployed using a Flask. Running the API file with the correct model pickle file in the Flask folder should provide a locally running sentiment analysis API.

It can take an arbitrary number of reviews and provides back the expected rating.

Recommender:

Built a very rough around the edges recommender system using User ID, Restaurant Name and rating review. It was run on a subset of the data set as the full dataset was far to large to be run using this methodology.

The model was created using cross fold validation on KNN having subtractd the mean value for each user from their recommendations.

Error Metric	Value
RMSE	1.03
MAE	0.75

These results are not great for a recommender system as it is saying it could be off on a recommendation by a whole scoring rank.

As a way to enrich the data and to use a more advanced modeling approach instead of a memory approach we could use features extracted from the Reviews by each customer on each restaurant and then use the features instead of the restaurants themselves.

Implementation of this would probably look like an API that accepted a User ID. If the user doesn't exist in the dataset then we would just generate a top N restaurants for that user based off of the aggregate dataset. This is to sidestep the cold start problem that new or low recommendation users would face.