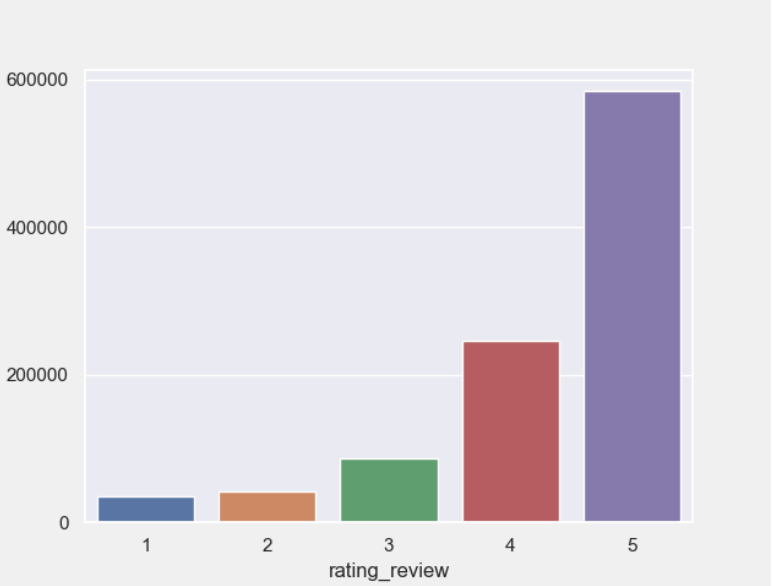
EDA:

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

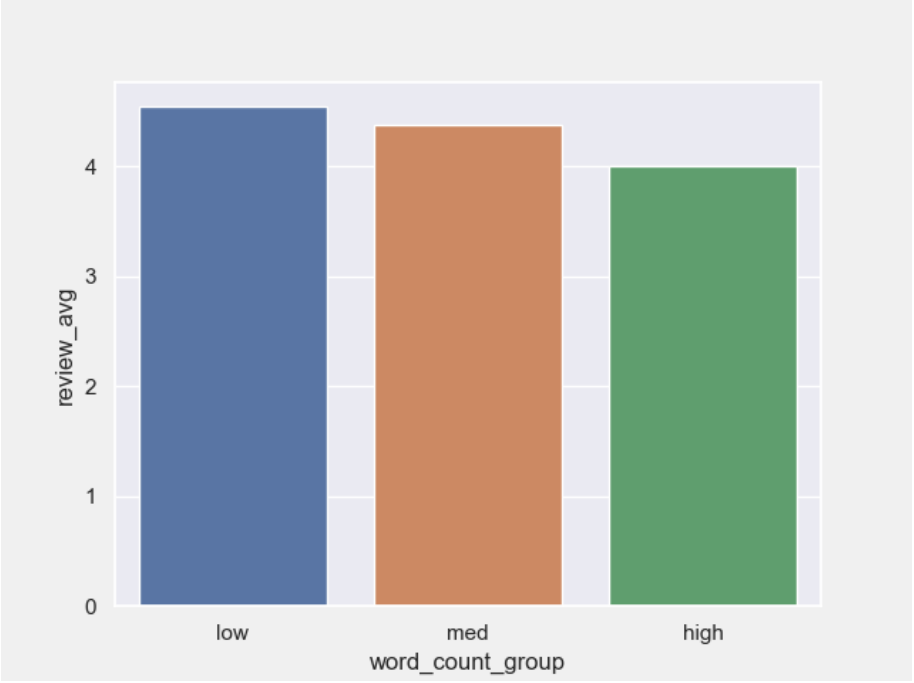
Any observations with blank reviews or scores were excluded from the analysis as there is no reliable way to infer what they would have said.



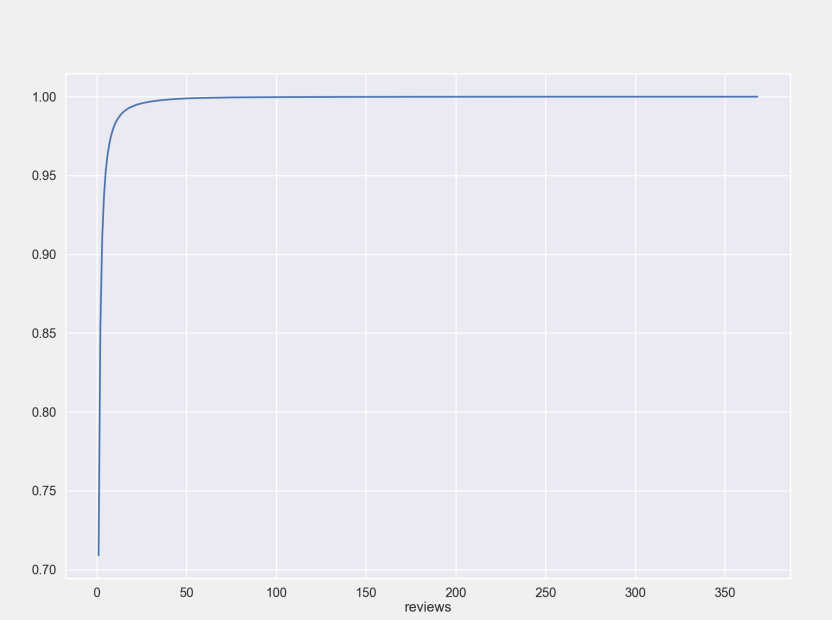
To start with some simple questions were asked of the data. The simplest being what is the distribution of our target variable.

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.

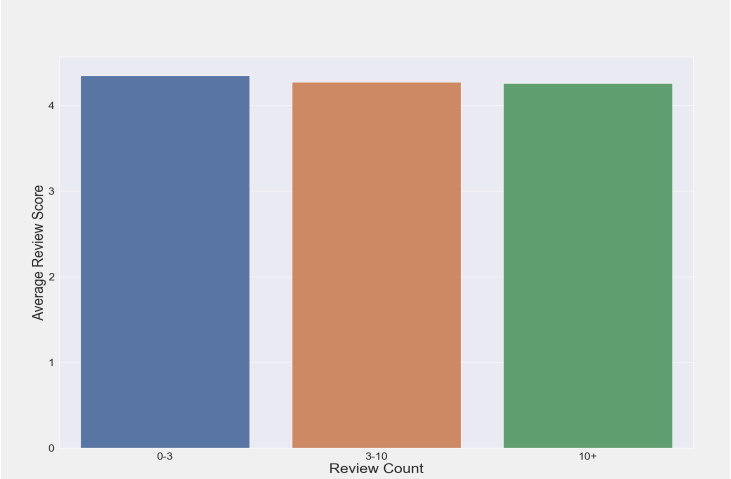


As demonstrated by the figure above the longer a review is the more likely it is to be negative. So while people rate more things highly they have a lot more to say about bad reviews.

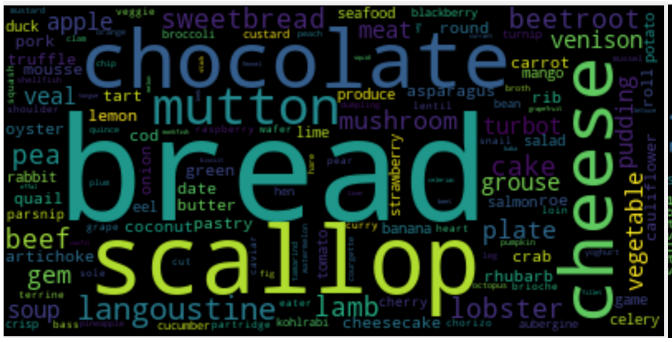


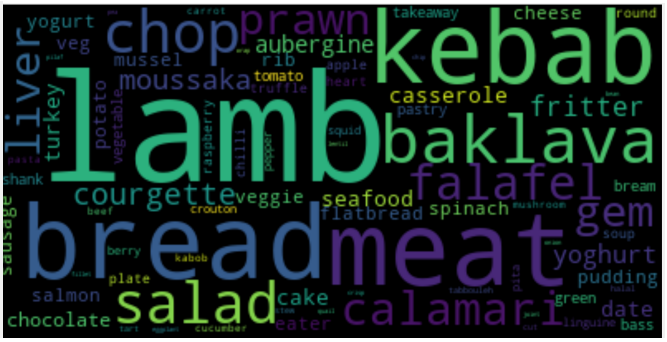
Next I took a look at the Reviewers themselves. And found out that a vast majority of users only have less than 10 review

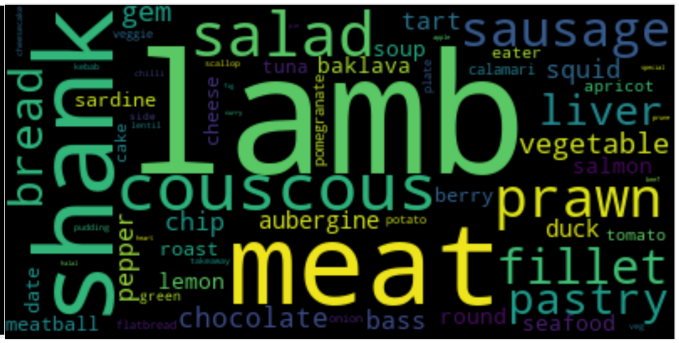
And looking at the scores users give. The more reviews given seems to show a trend to more honest or at least lower rated reviews. Even if the trend is very small.

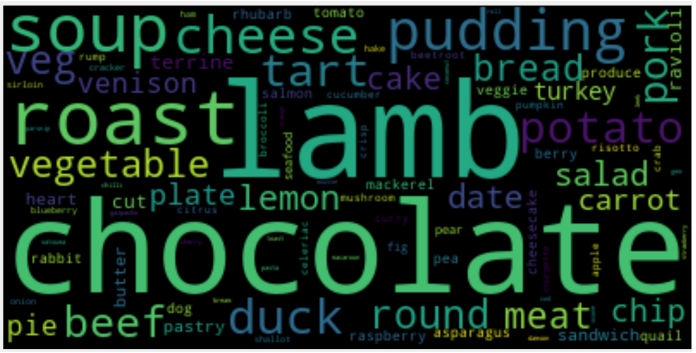


As for the restaurants, some feature analysis of the top and bottom restaurants with more than 500 reviews shows us that there is very little domination of restaurant brand in London. As you can see for the word clouds below the Top 5 cover a wide range of food and cuisine.

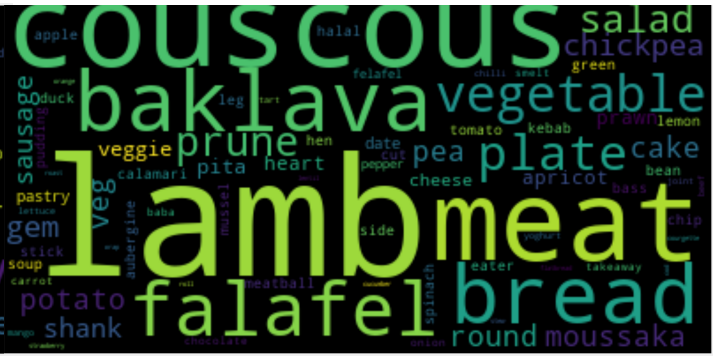








Just like the top rated restaurants there is no stand out worst cuisine in london either as they all offer different things, suggesting that in London you can always find somewhere good to eat in whatever style you want if you are willing to look.



Model:

I decided to use a grid search approach to a Random Forest Classifier as well as a multi nominal naive bays model. Both of thees are known to be good at multi-type classification problems.

Calibrated Model

The model us calibrated using a scikit learn pipeline object this is to make whichever of the model types end up being the most effective during the calibration process able to be seamlessly deployed to the api backend. As all it needs to do is load in the pickle file and run the predict function.

Naive Bayes

Reveiw Score	**precision**	**recall**	**f1-score**	**support**
1	0.6712904582	0.2227646813	0.3345203184	7169
2	0.3079019074	0.01341405508	0.02570811057	8424
3	0.3351009586	0.09404155458	0.1468668991	17471
4	0.3325838655	0.07461361527	0.1218832913	49562
5	0.6409458239	0.991721159	0.7786517912	116683
Accuracy Total			0.61596817	116683

Random Forest

Reveiw Score	**precision**	**recall**	**f1-score**	**support**
1	0.3186910006	0.3531873344	0.3350535927	7169
2	0.2504211117	0.05294396961	0.08740813327	8424
3	0.3802559415	0.08333810314	0.136713615	17471
4	0.3764309764	0.2030184415	0.2637760185	49562
5	0.6581836704	0.8970201315	0.7592625505	116683
Accuracy Total			0.5978806777	116683

From the small grid search I ran the Naieve Bayes model performed best and is what I deployed to my api. This is achieved by running the deploy.py script and specifying the version number you want to deploy. In production this can be achieved with something like ML Flow.

Both of the calibrated models however are much better at classifying the highly dominant case of 5 than any of the other labels. Which is dragging up the accuracy score. Further steps to take to improve the model could be things such as random understampling of the highly dominant classes (mainly score 5). Another option is to calibrate / call out to a deep learning model such BERT to do the heavy lifting of the model instead of calibrating a custom one ourselves.

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. There is a deployment script which removes the current model and replaces it with the model version specified.

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.

The code repo: Github