Task 4 & 5

Task 4 – Mining popular dishes

Problem statement

In this task, you will create a visualization showing a ranking of the dishes for a Yelp cuisine of your choice. You may use the [dish list we have provided](), the list based on your annotations from Task 3 (or a subset of that list), or any other list for other cuisines. You might find it more interesting to work on a cuisine for which you can recognize many dishes than one with only a few dish names that you recognize.

Approach 1

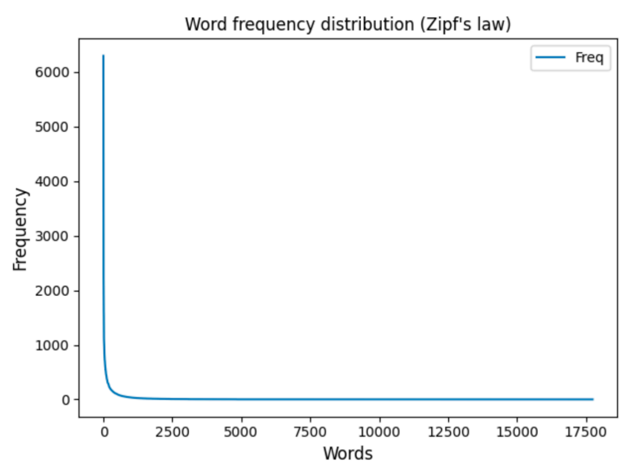
As suggested, the simplest approach can be to simply count how many times a dish is mentioned in all the reviews of restaurants of a particular cuisine.

With source of truth as the annotated list provided by the course here.

'student_dn_annotations.txt' file has 2085 dish names provided. I used all the reviews gathered (from task 3) for "Indian" cuisine and calculated the word frequency by removing words based in 'stopwords.txt'. To no surprise, the frequency pattern follows "Zip's Law".

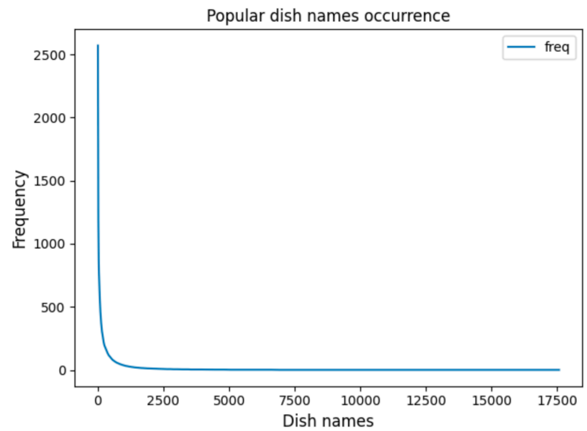
Based on unigram word generation

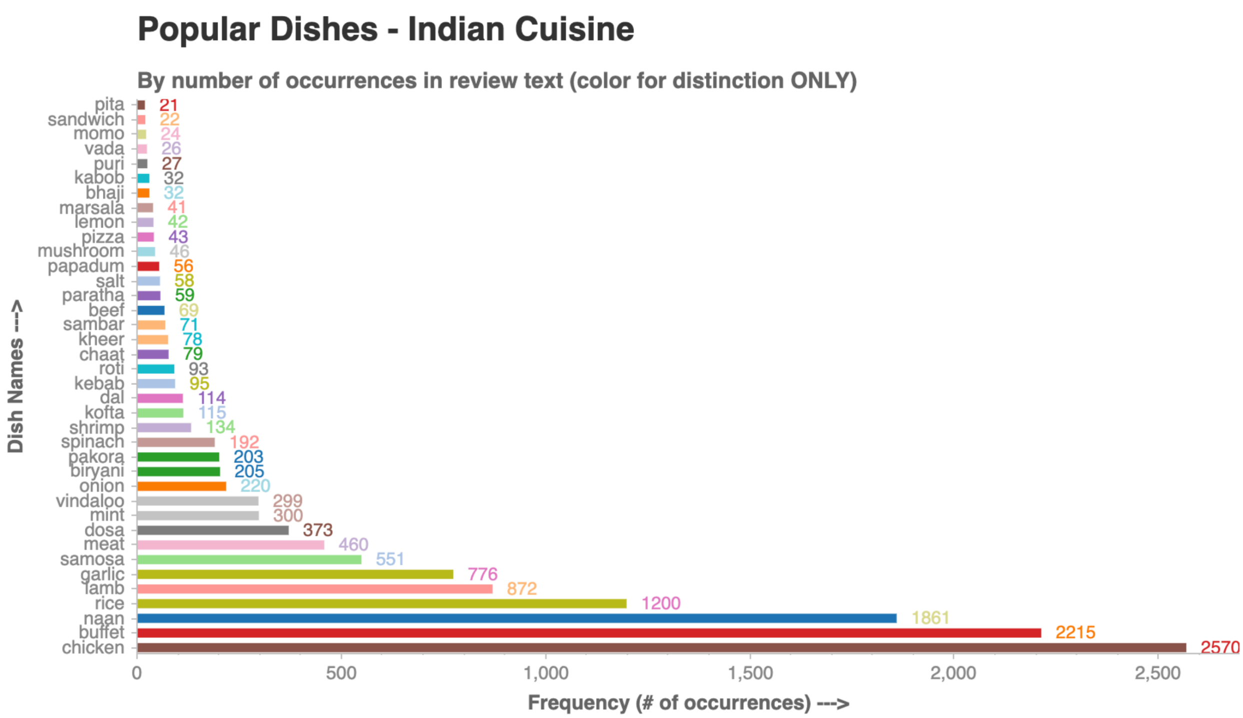
```python
all_text = ''
with open(input) as fh:
    for line in fh:
        line = line.strip()
        all_text = all_text + line

doc = metapy.index.Document()
doc.content(all_text)
tokens = build_collection(doc, 'stopwords.txt')

all_words = nltk.FreqDist(tokens)
string = ''
for word, frequency in all_words.most_common():
    string += u'{},{}\n'.format(word, frequency)

write_to_file(output, 'word_frequency_reviews.csv', string)
```



Next thing is to apply filtering of words as a "look up" from "word_frequency.csv" and gather relevant frequency of given words in 'student_dn_annotations.txt'. There are dishes from other cuisines as well, so obviously there will be dish names which will not have any corresponding frequency in the corpus I used (Indian reviews text file).

```
def look_up(search, search_db, output_file):
    # convert search word per line into an array
    search_array = []
    with open(search) as fh:
        for line in fh:
            line = line.strip()
            search_array.append(line)

    column = ['word', 'freq']
    X = pandas.read_csv(search_db, names=column)
    Y = X[X['word'].isin(search_array)]
    Y.to_csv(output_file, index=False, header=False)
    ax = Y.plot(legend=True, title='Popular dish names occurrence')
    ax.set_xlabel('Dish names', fontsize=12)
    ax.set_ylabel('Frequency', fontsize=12)
    plt.show()
```



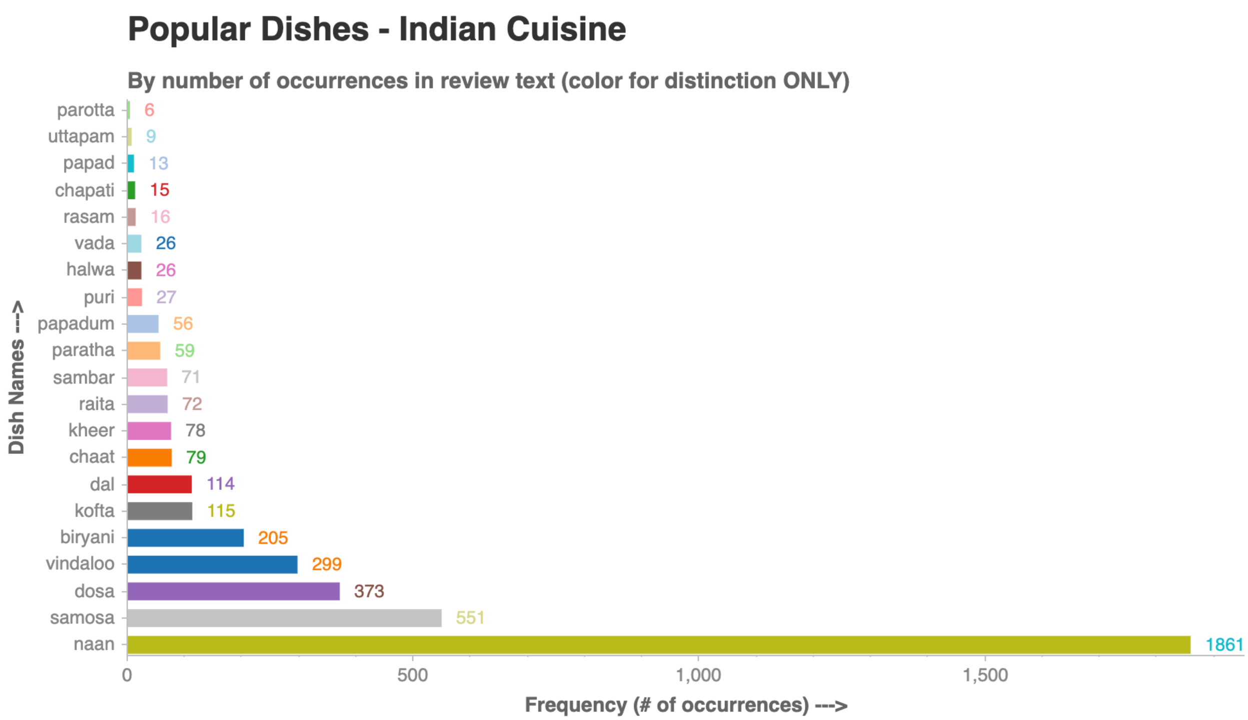


Analysis

With a given annotation of mixed dish names from different cuisine, it's motivating to see following dish names (in graph above) as output of this solution. Note, I have dropped the dish for frequency less than 20. So chicken, naan, rice, lamb, samosa, dosa, coming on top makes a lot of sense as I can understand from traditions in Indian food culture. Buffet is a way of catering, garlic is condiment/spice, so since we have such weak annotations, so is some of dish names shown in visualization.

With source of truth that I gathered as a list of dishes that I generated for "Indian" cuisine

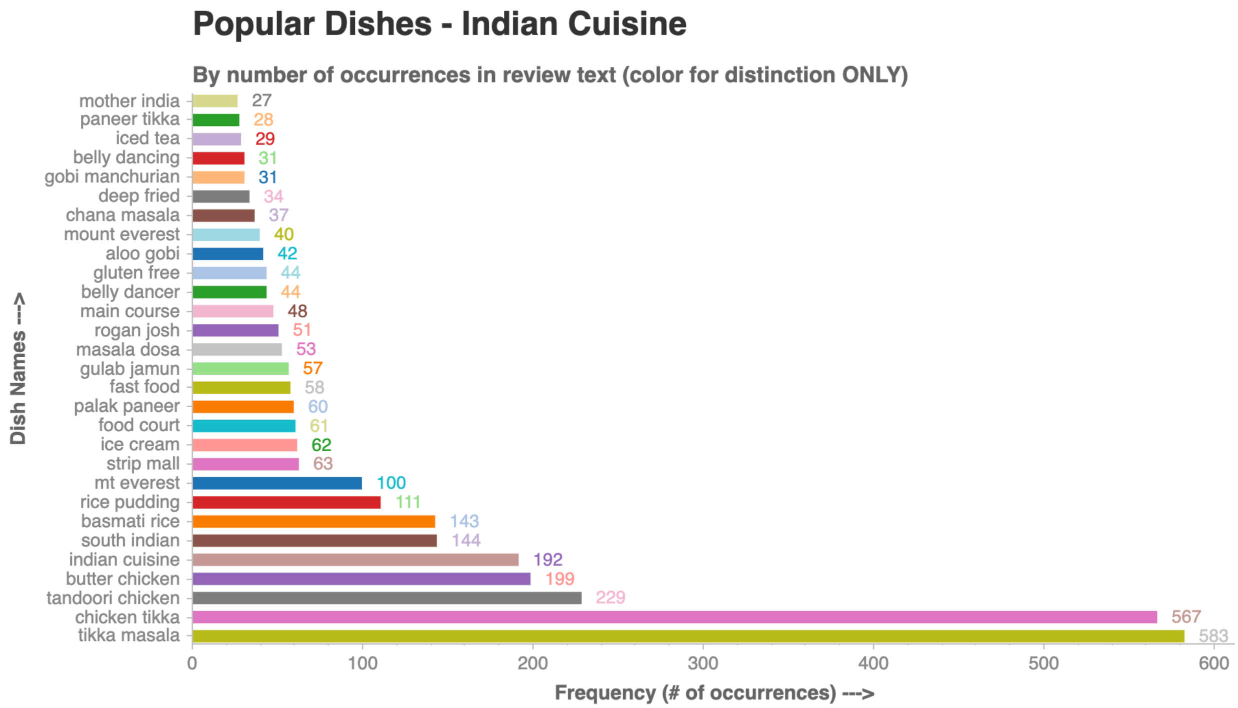
Here I used "indian_dishnames_task3.txt" file from task 3 which has dish names from wiki also. Note this is also based on **<u>unigram</u>** and I have ignored frequency 5 and below. Here again naan, samosa, dosa are from top few. I was surprised why "biryani" was missing that's because we used unigram approach and the first search criteria, biryani was mentioned as bigram words "mutton biryani"



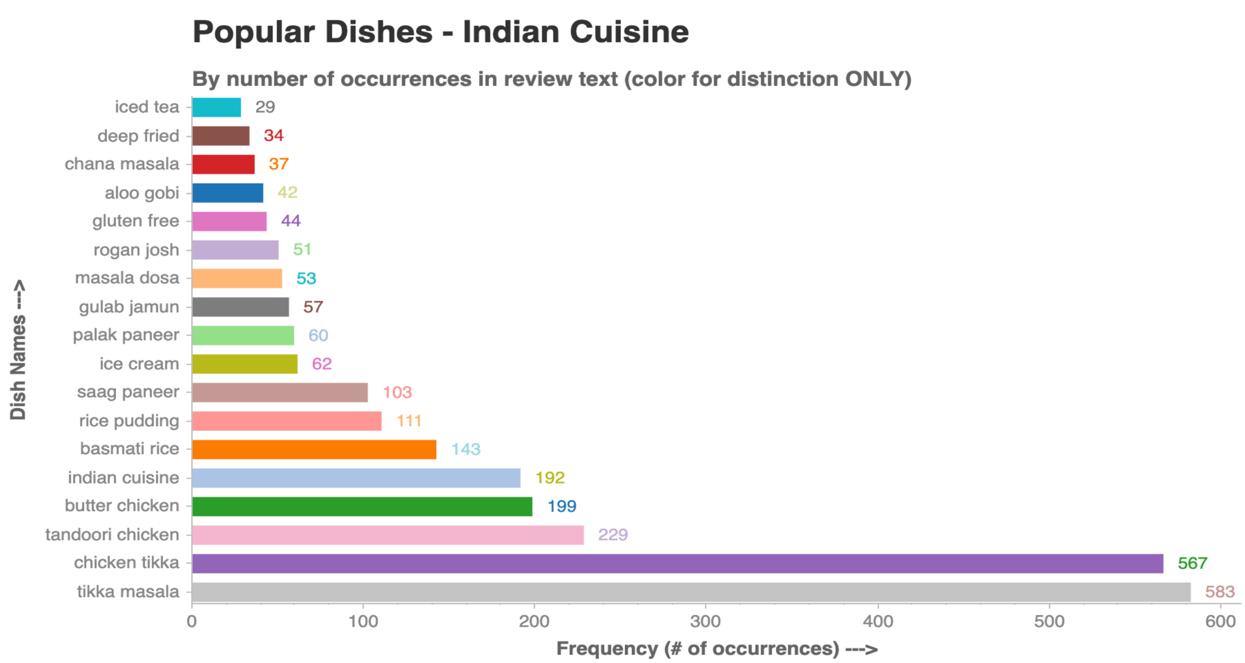
Bigram word generation with my own annotated cuisine data

Dish names are more likely to be at least 2 words. So just to experiment around, I have used nltk package and specifically removed words/chars like 'i' , 'the', 'we', 'this', ''d', 'don't', 'don't', and number 0 to 9, apart from 'stop_words.txt' (this is due to my observance on such occurrences) and also did lower case. This improved overall quality of test extraction.

Then used this to apply filter just like explained above. Below graph shows popularity based on bigram on 'student_dn_annotation.txt'. Tikka masala, chicken tikka, tandoori chicken and butter chicken are some of very famous Indian dishes indeed. And this is much better and informative result compared to unigram model. Single word is hard to interpret, and as we have learnt through experience more words help develop the context more clearly.



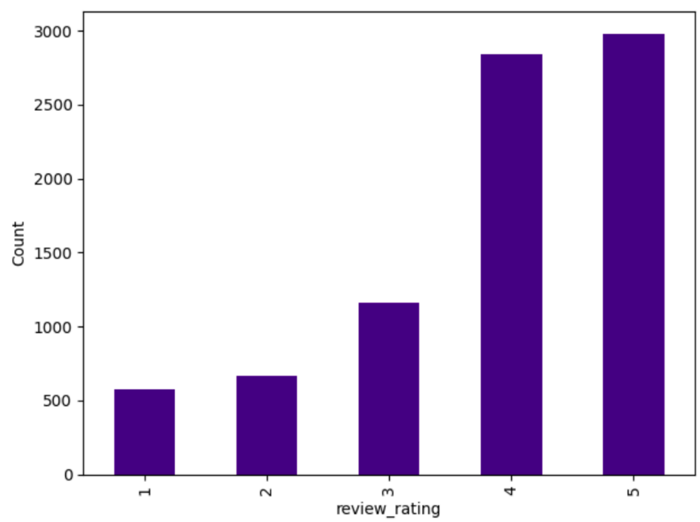
Below is the result using my own set of annotated popular dishes from task 3. The method that I applied remains same as explained in above sections. It's amazing to see similar top dishes as above experiment. I can definitely conclude quality extraction of information creating knowledge base.



Approach 2

Another approach I have tried is to understand the dishes spread based on the ratings of individual reviews, for "Indian" cuisines only (regardless of the restaurant ID). Here I have taken the data as explained below and performed sentiment analysis.

I refactored the code provided in 'py27_processYelpRestaurants.py' to dig out ONLY "Indian" cuisine as category and ONLY to find restaurant IDs related to this cuisine. And then I looped over the reviews and found all the reviews and its respective rating and dumped all in flat structure as "restaurant_id" , "restaurant_rating", "review_text" and "review_rating" which are ONLY for "Indian" cuisine. *<u>This is good help for Task 5</u>*.




```
Shape of the data set: (8228, 4)
       restaurant_rating  review_rating
count        8228.000000    8228.000000
mean            3.542902       3.849538
std             0.531476       1.197508
min             1.000000       1.000000
25%             3.000000       3.000000
50%             4.000000       4.000000
75%             4.000000       5.000000
max             5.000000       5.000000
review_rating
1     576
2     664
3    1163
4    2844
5    2981
```

Then I did simple sentiment analysis just based on individual reviews with parameters as follows. Naïve Bayes classifier with 8000 data set over 162000 words feature sets. *Below from package:*

A Naive Bayes classifier. Naive Bayes classifiers are paramaterized by two probability distributions:
- P(label) gives the probability that an input will receive each label, given no information about the input's features.
- P(fname=fval|label) gives the probability that a given feature (fname) will receive a given value (fval), given that the label (label).
If the classifier encounters an input with a feature that has never been seen with any label, then rather than assigning a probability of 0 to all labels, it will ignore that feature. The feature value 'None' is reserved for unseen feature values; you generally should not use 'None' as a feature value for one of your own features.

Once the model was trained, I started looking for classifications for each of the "bigram" dish names that I generated from my own annotation (I had saved the bigram frequency file as 'indian_dishnames_task3_freq_bigram.csv'). So instead of just random color for representation I hope to view ***"frequency along with sentiments".***

Tried below classifiers and conclusion

Pretty much everything is below 46% accuracy even after so much of data cleaning. Logistic regression gave about 46.55% accuracy, so went with that. This could be because I did not categorize based on positive and negative, but rather went with larger scale of 1 to 5 as classification on sentiments.

- LinearSVC_classifier
- SVC_classifier
- SGDClassifier_classifier
- LogisticRegression_classifier
- BernoulliNB_classifier
- MNB_classifier
- Naïve Bayes

So, I tried with TfidfVectorizer, created a pipeline with 1200 best and used random forest algorithm.
And the precision was looking good around 0.7 but to my surprise no matter what classification algorithm I used, I ended up getting classified as "4". Then later I realized that due to cutting down the selected dishes to limited high frequency number I could be getting all "4" classified. *So, there is no point of generating visualization as it will be same classification color* ☹

Glimpse of code and inference

```
from sklearn.utils import shuffle
df = shuffle(dataset)
stemmer = nltk.PorterStemmer()
words = stopwords.words("english")
df['cleaned'] = df['review_text'].apply(
    lambda x: " ".join([stemmer.stem(i) for i in re.sub("[^a-zA-Z]", " ", x).split() if i not in words]).lower())

vectorizer = TfidfVectorizer(min_df=3, stop_words="english", sublinear_tf=True, norm='l2', ngram_range=(1, 2))
X = df['cleaned']
Y = df['review_rating']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.55)
# instead of doing these steps one at a time, we can use a pipeline to complete them all at once
pipeline = Pipeline([('vect', vectorizer),
                     ('chi', SelectKBest(chi2, k=1200)),
                     ('clf', RandomForestClassifier())])
# fitting our model and save it in a pickle for later use
model = pipeline.fit(X_train, y_train)
with open('RandomForest.pickle', 'wb') as f:
    pickle.dump(model, f)
ytest = np.array(y_test)
print(classification_report(ytest, model.predict(X_test)))
print(confusion_matrix(ytest, model.predict(X_test)))

test_dataframe = pandas.read_csv('outputpath/indian_dishnames_task3_freq_bigram.csv', names=['word', 'freq'])
abc = model.predict(test_dataframe['word'])
```

```
              precision    recall  f1-score   support

           1       0.69      0.28      0.39       326
           2       0.45      0.04      0.08       354
           3       0.37      0.18      0.24       634
           4       0.43      0.58      0.49      1559
           5       0.59      0.69      0.64      1653

    accuracy                           0.50      4526
   macro avg       0.51      0.35      0.37      4526
weighted avg       0.50      0.50      0.47      4526

[[  90   11   44  131   50]
 [  26   15   81  177   55]
 [   7    6  114  419   88]
 [   4    0   46  907  602]
 [   3    1   21  487 1141]]
```

Task 5 – Restaurant recommendation

In this task, your goal is to recommend good restaurants to those who would like to try one or more dishes in a cuisine. Given a particular dish, the general idea of solving this problem is to assess whether a restaurant is good for this dish based on whether the reviews of a candidate restaurant have included many positive (and very few negative) comments about the dish. You may choose a target dish or a set of target dishes from the list of "popular dishes" you generated from Task 4 or, otherwise, choose any dishes that have been mentioned many times in the review data (the more reviews you have for a dish, the more basis you will have for ranking restaurants).

Approach 1

Gather all reviews of restaurant (for a dish) and take average of review rating. And then rank restaurants name.

Here I created separate solution in 'popular_restaurant_by_dish.py'.

Performed below steps:

- Used file generated while I was trying to do sentiment analysis, a file which is only for 'Indian cuisine' with almost all what I need. '**flat_review_per_restaurant_with_rating.csv**'
- Set dish search criteria "Chicken Tikka"
- Did data cleaning
- Got unique list of restaurants for 'Indian cuisine'
- Gathered set of reviews for each restaurant which mentions "chicken tikka", calculated the average rating where this dish name was mentioned and took the reviews count too.

There are 2 ways I generated the visualization for recommendation, one is by average rating and another by number of reviews. One could argue pros and cons of both approaches. But as a user I think I would love to see such insights which can help me take decision from multiple perspective. Skip the table below (its long) but interesting information and self explanatory.

Restaurant	Average rating	Number of reviews
Noor Indian Takeaway	5	1
The Cholas	5	1
Haveli Restaurant	5	1
10-to-10 In Delhi	5	1

Pak Afghan Halal Restaurant & Catering	5	1
Aachi Southindian Kitchen	5	1
Kebabish Original	5	1
Mother India's Cafe	5	3
Kismot	5	1
Aromas at Island Restaurant	5	1
Haweli Indian Grill & Bar	5	2
Kebab Mahal	4.8	5
Indus Village	4.666666667	3
Nandini Indian Cuisine	4.666666667	3
Flavor of India	4.6	5
Mount Everest India's Cuisine	4.513513514	37
Madras Ananda Bhavan	4.5	2
Delhi Indian Cuisine	4.4	5
OM Restaurant	4.375	8
Star of India	4.357142857	14
India Garden	4.333333333	3
Indian Paradise 2	4.333333333	3
Swad	4.285714286	7
Dhaba Indian Bistro	4.285714286	14
Khyber Halal	4.25	16
Royal Taj	4.222222222	27
Tandoori Times 3 Indian Bistro	4.181818182	11
Guru Palace	4.176470588	17
Maharaja Restaurant E Pt Plaza	4.166666667	6
Maharaja Restaurant	4.166666667	6
India Oven	4.130434783	23
Copper Kettle	4.125	8
India Masala	4.111111111	9
Indian Delhi Palace	4.066666667	15
Tandoori Times 2 Indian Bistro	4.0625	16
The Dhaba	4.03125	32
Taste of India	4	1

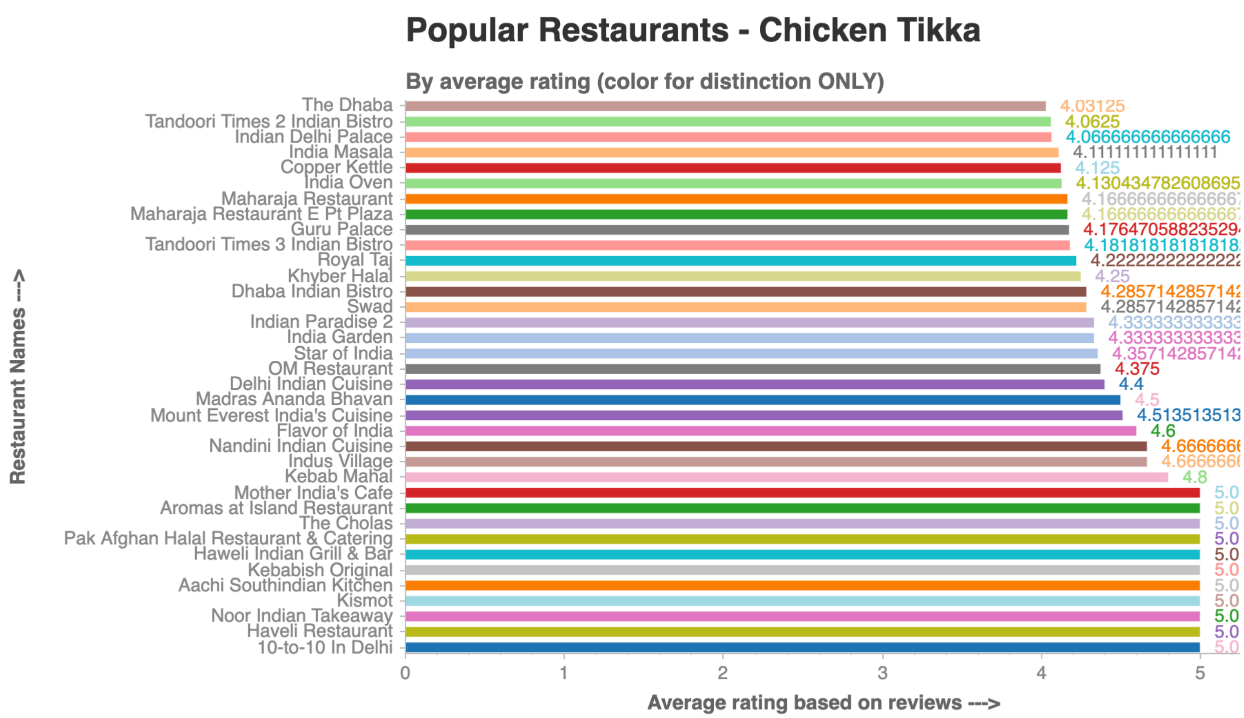
Taj Indian Restaurant	4	5
New India Bazaar	4	9
Al Hamra	4	4
Imans	4	1
Curry Garden	4	3
Tanjore	4	1
Mirch Masala	4	2
Iman's Grill	4	1
The Clay Oven	4	1
Mint Indian Bistro	4	30
India's Grill	4	4
Samosa Factory	4	13
Swagat India	4	8
Pastries N Chaat	4	1
Zaidi's Grill	4	1
Mezbaan South Indian Restaurant	4	2
Sutra Fine Indian Cuisine	4	2
India Grill	4	3
Omar Khayyam	4	1
Tamba Indian Cuisine & Lounge	4	18
Curry in the Box	4	2
Shezan Indian Cuisine	4	2
India Palace	3.957142857	70
Taj Palace	3.931034483	29
Delhi Palace Cuisine of India	3.923076923	13
Shalimar Restaurant	3.916666667	12
Flavors of India	3.866666667	15
Saffron Flavors of India	3.8125	16
Indian Paradise	3.782608696	23
Southern Spice	3.777777778	9
Maharani Restaurant	3.666666667	6
Minerva Indian Cuisine	3.666666667	3
Gaylord India Restaurant	3.666666667	6
Taj Mahal Indian Cuisine	3.666666667	9
Chennai Chettinaad Palace	3.666666667	6
Bawarchi Indian Cuisine	3.625	8

India Masala Bar & Grill	3.588235294	17
Origin India Restaurant & Bar	3.583333333	24
Jewel of the Crown	3.545454545	11
Khushi's	3.5	2
Kabab Palace	3.5	4
Bollywood Grill Indian Cusine	3.5	2
Mayuri Palace	3.5	6
Bay Leaf Cafe	3.461538462	13
Mantra Masala	3.454545455	22
Namaste Indian Cuisine	3.4	5
Curry Corner	3.387096774	31
Tandoori Times Indian Bistro	3.35	20
Karaikudi Palace	3.333333333	6
Kabob n Kurry	3.333333333	6
Pasand	3.333333333	3
Kohinoor Cuisine of India	3.25	4
Gandhi India's Cuisine	3.222222222	9
Copper Kettle-Salads Balti & Taandoori Grill	3.2	5
Royal India Bistro	3.2	5
Bombay Spice Grill	3.166666667	18
Masala Bay Fine Indian Dining	3	1
Shamoli Thai & Indian Cuisine	3	1
Hurry 4 Curry	3	1
Curry House	3	3
NASHA Indian Cuisine	3	3
Sai India Curry	3	1
Passage To India	3	8
Chutney's Indian Cuisine	3	5
Indian Curry Bowl	2.875	8
India Gate	2.833333333	6
Tandoori Village	2.5	4
Indian Maharaja Palace	2.5	2
Taza Indian Kitchen	2.333333333	3

Curry Leaf Indian Grill	2	1
Tadka Sizzles	2	1
Saffron	1.5	4
Pyaar India Restaurant	1	1
Kabob N More	1	1
Zest of India	1	1

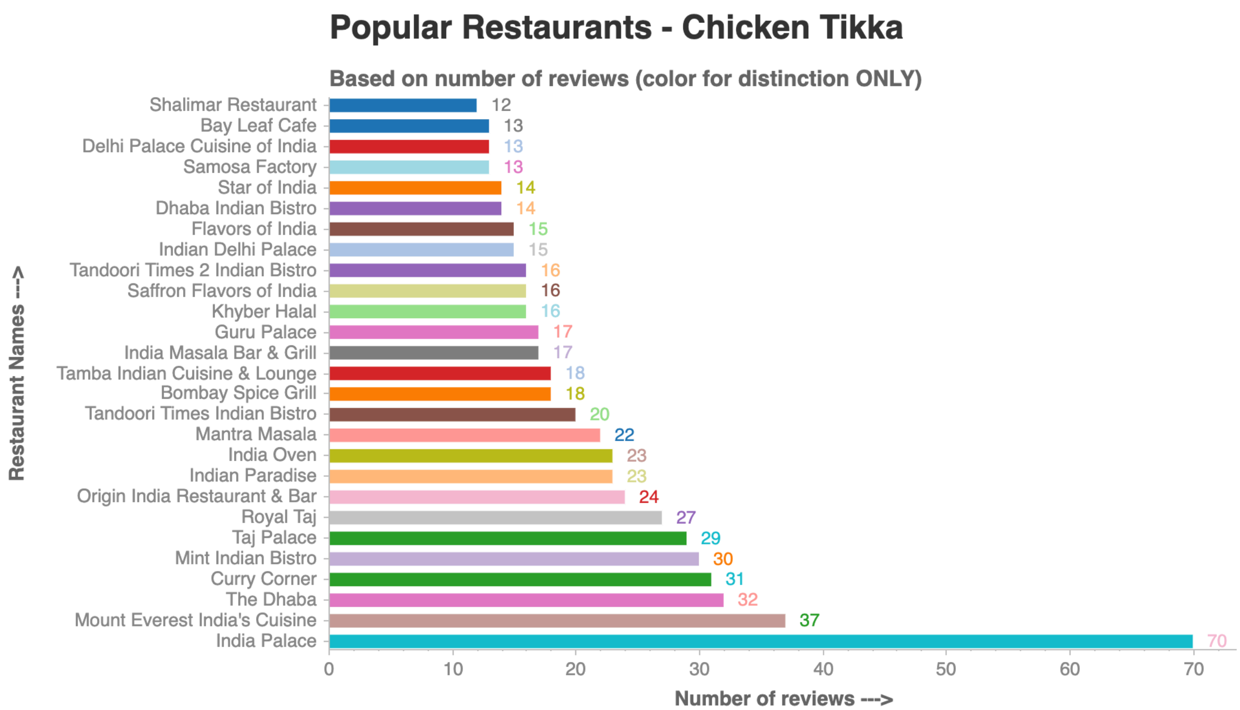
Reviews rating greater than 4

Here almost I have got more than 10 restaurants for 5-star rating (which is average of multiple reviews). But the catch is I don't feel like it's a justified approach since we have a smaller number of reviews for some restaurants and just by 1 or 2 giving high or low rating, the information gets biased.



Reviews greater than 11 occurrences

This approach makes a better sense to me as here the number of reviews is taken into account. Now let's take a closer look at the top candidates by average rating and number of reviews, we can notice as I explained earlier there could be potential bias because of smaller number of reviews, the results are different. But when both approaches are taken and the users are warned upfront on the assumptions, these insights could be very useful.



Approach 2

One to do with search by gathering all reviews of each restaurant as single document and then rank based on search on a dish.

Steps taken:

- Used the file generated from above approach and took all the reviews per restaurant for that dish name. (another approach could also be without letting any bias of dish name on reviews and then rank)
- Then used metapy library as explained below
- Used bigram and OkapiBM25 algorithm for scoring

At the heart of MeTA lies its indexes. Every piece of data that MeTA's algorithms operate on must first reside in an index. Create inverted_index and used the same to create a search engine. (https://meta-toolkit.org/search-tutorial.html)

Analyzer (config.toml):

```
prefix = "."

dataset = "dataset"
corpus = "line.toml"
index = "idx"

[[analyzers]]
method = "ngram-word"
ngram = 2
filter = [{type = "icu-tokenizer"}, {type = "lowercase"}]
```

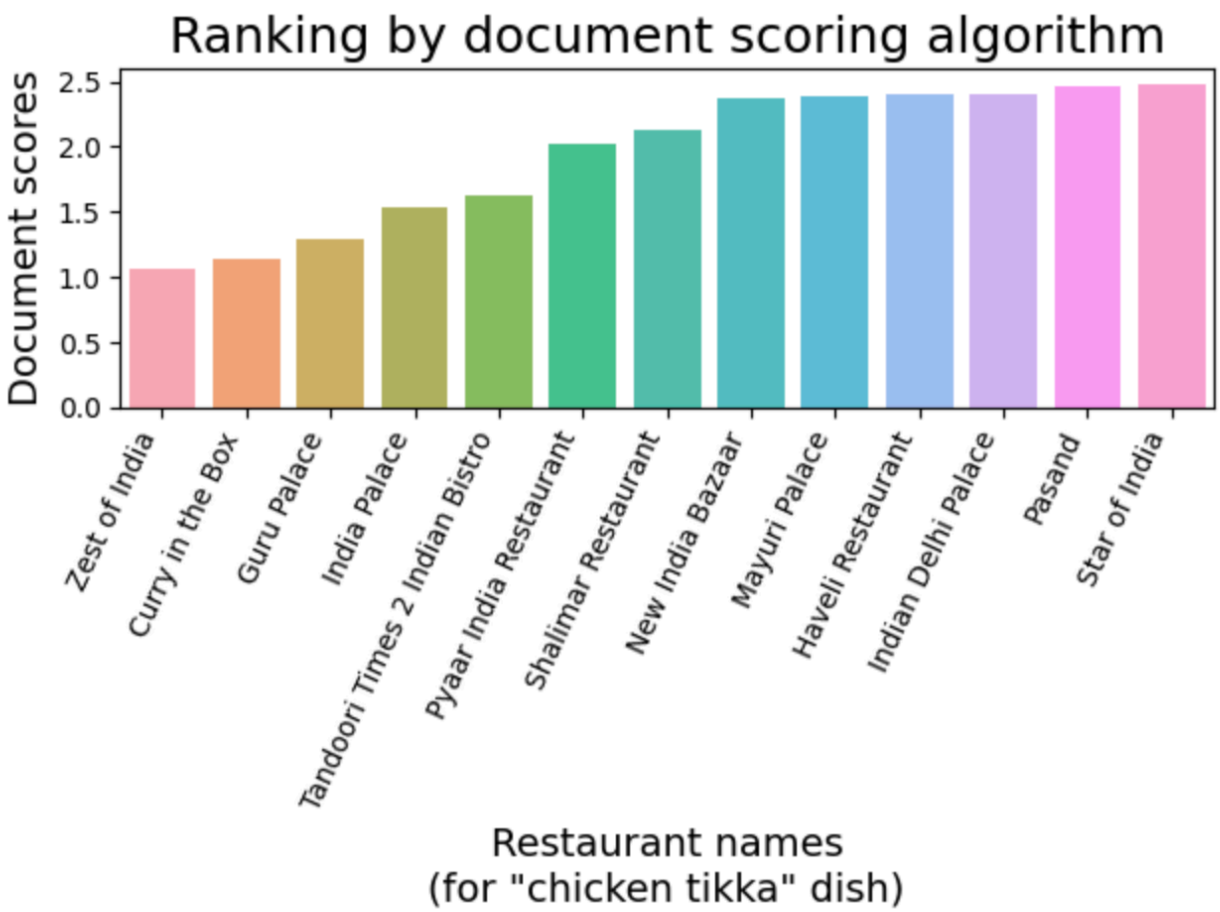
line.toml

```
type = "line-corpus"
data = {name = "content", type = "string"}
encoding = "utf-8"
store-full-text = true
```

Once the index was created, I searched for “chicken tikka”

```
query.content('chicken tikka')
top_docs = ranker.score(idx, query, num_results=100)
```

This gives a tuple of index and corresponding score for the document. I saved this into a file (There is a struggle with this library as it works with python 2.7). Then called visualization to create the knowledge as below. Pasand, star of India, Indian Delhi Palace are few top scorers. But it’s interesting to corelate with approach 1 that I took. I think this is more admissible approach as it follows proper principles of TFIDF, document penalizing for length and such approaches with BM25 and okapi.



Feel free to explore my GitHub link for this solution

https://github.com/atif-github-venture/data-mining-capstone