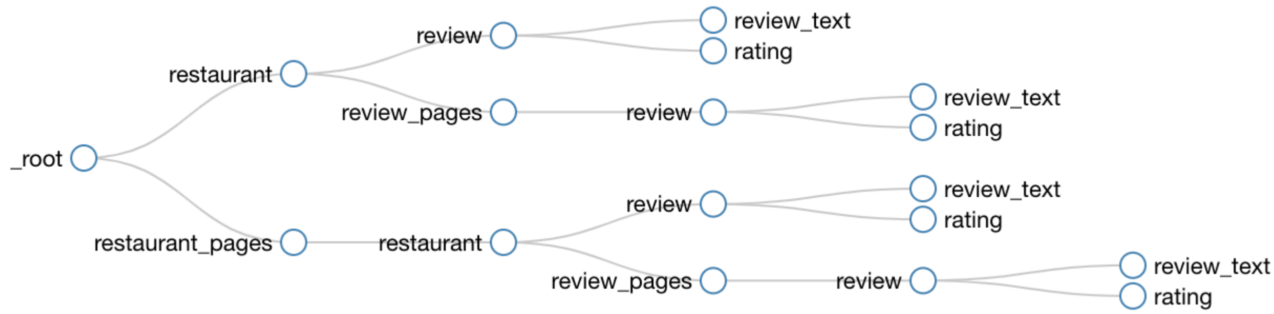
Finding the best Japanese restaurant in Austin

Task A. Scrape Yelp and collect reviews

Using a Web Scrapper extention for Google Chrome, I collected a total number of **12403 reviews** related to **Japanese restaurants** in the Austin area. Below is the graph selector graph that was used.



Based on the user uploaded reviews for every restaurant, I created a dataset with the below data:

1. Name of the restaurant
2. Review text
3. Rating of the resturant by the reviewer

Task B. Word frequency analysis

As a first step, I analysed the reviews in regards to word frequencies. This is the most important part of every context analysis, as it allows for the extraction of features through the text.

In [1]:
```
import pandas as pd
import numpy as np

from nltk import pos_tag
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from collections import Counter
```

Using the collected reviews, we imported the data and separated the column related to reviews.

In [2]:
```
# importing the data
yelp_data = pd.read_csv('ja3.csv')
```

In [3]:
```
# creating a table with all the restaurant reviews
rest_review = yelp_data.loc[:,{'restaurant','review_text'}].dropna()
rest_review.head()
```

Out[3]:

	review_text	restaurant
0	Awesome tastes! Especially like the banana pudd...	Kemuri Tatsu-ya
1	Heard about the Fukumoto hype so figured it wa...	Fukumoto Sushi & Yakitori
2	There were only 2 people to do the hibachi on ...	Nagoya Steak and Sushi
3	I'm from California, land of the fresh sushi. ...	Dawa Sushi
4	Amazzinnggg staff and amazing food!!!!!! Love ...	Soto - South Lamar

In [4]:
```
# selecting the reviews from the data
reviews = pd.DataFrame(yelp_data.iloc[:]['review_text'])
reviews.shape
```

Out[4]: (25315, 1)

Next, I removed the empty rows, which where already identified as a result of the way the scrapper worked and did not effect the analysis.

In [5]:
```
# keeping only the valid non empty reviews
unique_reviews = reviews.apply(lambda x: pd.Series(x.dropna().values
unique_reviews.shape
```

Out[5]: (12403, 1)

Using this final list of reviews, I collected them all into a single string and counted the occurence of each word. In addition words were tokenized while stop words were removed since they were of no interest to me in regards to the subject analysis.

In [6]:
```
# combining all the reviews into a single string
reviewz = []
for i in range(len(unique_reviews)):
    # making all text into lower case and appending to a single list
    reviewz.append(unique_reviews.loc[i][0].replace('\n', '').lower(
```

In [7]:
```
all_reviews = ''.join(reviewz)
```

In [8]:
```
# counting the occurencies of words and tokenzing them
tokens = word_tokenize(all_reviews)

# stemming the words
# stem_tokens = [ps.stem(w) for w in tokens]
stem_tokens = tokens

# adding pos tag to the words and counting occurencies
tokens_pos = pos_tag(stem_tokens)
wordcount = Counter(tokens_pos)
```

In [9]:
```
# sorting the words based on their frequency
word_list = sorted(list(wordcount.items()), key = lambda w: -w[1])

# keeping only words with length greater than 2
word_list = [word_list[i] for i in range(len(word_list)) if len(word

word_list[:10]
```

Out[9]:
```
[(('the', 'DT'), 70091),
 (('and', 'CC'), 47108),
 (('was', 'VBD'), 28019),
 (('for', 'IN'), 15900),
 (('but', 'CC'), 12283),
 (('you', 'PRP'), 11348),
 (('with', 'IN'), 10661),
 (('this', 'DT'), 9889),
 (("n't", 'RB'), 9851),
 (('sushi', 'NN'), 9464)]
```

In [10]:
```
# introducing stop words and creating a list of them
import nltk
nltk.download('stopwords')

stoplist = nltk.corpus.stopwords.words('english')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/thomas/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [11]:
```
# filetering out stop words
no_stopword_list = []

for i in range(len(word_list)):
    if word_list[i][0][0] not in stoplist and len(word_list[i][0][0]
        no_stopword_list.append(word_list[i])

# filtering out by pos tags
pos_tags = ['NN', 'NNP', 'JJ']
no_stopword_list = [no_stopword_list[i] for i in range(len(no_stopwo
```

In [12]:
```
no_stopword_list
```

```
(('great', 'JJ'), 5430),
(('service', 'NN'), 4603),
(('roll', 'NN'), 4152),
(('time', 'NN'), 3856),
(('order', 'NN'), 3009),
(('restaurant', 'NN'), 2863),
(('menu', 'NN'), 2636),
(('fresh', 'JJ'), 2336),
(('happy', 'JJ'), 2288),
(('delicious', 'JJ'), 2258),
(('rice', 'NN'), 2234),
(('hour', 'NN'), 2173),
(('nice', 'JJ'), 2151),
(('lunch', 'NN'), 2113),
(('little', 'JJ'), 1962),
(('spicy', 'NN'), 1880),
(('austin', 'NN'), 1795),
(('chicken', 'NN'), 1660),
(('staff', 'NN'), 1652),
(('japanese', 'JJ'), 1593),
```

Task C. Key features of success

Based on the above word frequences, I decided to analyse the collected even further data and group words together that refer to the same issue. In those terms, the following were identified as the most important issues related to japanese restaurants in Austin:

1. Service
2. Food
3. Price
4. Location

And to implement this features, I made relevant **replacements** to the data (presented below for reference). One important aspect of the analysis was that the word replacements related to food and were chosen in order to include words related to japanese cuzine (e.g. shushi, miso, nigiri, etc.) in order to "reward" reviews focusing on japanese cuzine. The best japanese restaurant in Austin needs to be focusing on japanese delights.

In [13]:
```
service = ['place', 'order', 'staff', 'table', 'friendly','waitress'
           'waiter', 'wait', 'clean', 'atmosphere', 'presentation',
           'music', 'seating']

food = ['sushi', 'roll', 'menu', 'rice', 'lunch', 'delicious', 'fres
        'fish', 'flavor', 'chicken', 'soup', 'bowl', 'dinner', 'shri
        'sashimi', 'fried', 'teriyaki', 'miso', 'crab', 'beef', 'egg
        'fish', 'avocado', 'eel', 'gigner', 'steak', 'meat', 'appeti
        'broth', 'salad', 'fish', 'tempura', 'dish', 'portion', 'pla
        'ginger', 'seafood']

price = ['expensive', 'worth', 'cheap', 'cost', 'money', 'dollar', '

location = ['area', 'spot', 'town', 'reasonable', 'parking', 'downto
```

Implementing the same word frequency analysis after the replacements, I had the below results:

In [14]:
```
word_reviews = []

for r in reviewz:
    #if len(set(t.split(' ')).intersection(replace_check)) > 0:
    for word in service:
        if word in r:
            r = r.replace(word, 'service')
    for word in food:
        if word in r:
            r = r.replace(word, 'food')
    for word in price:
        if word in r:
            r = r.replace(word, 'price')
    for word in location:
        if word in r:
            r = r.replace(word, 'location')
    word_reviews.append(r)
```

In [15]:

```
w_reviews = ''.join(word_reviews)

w_tokens = word_tokenize(w_reviews)

w_tokens_pos = pos_tag(w_tokens)
w_wordcount = Counter(w_tokens_pos)

w_word_list = sorted(list(w_wordcount.items()), key = lambda w: -w[1
w_word_list = [w_word_list[i] for i in range(len(w_word_list)) if le

w_no_stopword_list = []

for i in range(len(w_word_list)):
    if w_word_list[i][0][0] not in stoplist and len(w_word_list[i][0
        w_no_stopword_list.append(w_word_list[i])

w_no_stopword_list = [w_no_stopword_list[i] for i in range(len(w_no_

w_no_stopword_list
```

Out[15]:

```
[(('food', 'NN'), 70395),
 (('service', 'NN'), 27879),
 (('good', 'JJ'), 7518),
 (('great', 'JJ'), 5430),
 (('location', 'NN'), 3928),
 (('time', 'NN'), 3856),
 (('restaurant', 'NN'), 2866),
 (('price', 'NN'), 2652),
 (('happy', 'JJ'), 2288),
 (('hour', 'NN'), 2173),
 (('nice', 'JJ'), 2154),
 (('little', 'JJ'), 1965),
 (('austin', 'NN'), 1856),
 (('japanese', 'JJ'), 1593),
 (('quality', 'NN'), 1546),
 (('sauce', 'NN'), 1539),
 (('everything', 'NN'), 1452),
 (('first', 'JJ'), 1417),
 (('much', 'JJ'), 1364),
```

In [16]:

```
# replacing reviews in the restaurant - review table
rest_review['review_text'] = pd.DataFrame(word_reviews)
```

Task D. Cosine Similarily Analysis

Next I used sklearn metrics, in order to identify cosine similarities between the features I have chosen (service, food, price and location) and the collected reviews. It is like running a document query with words = service, food, price and location, and finding documents which are the best matches for the query words. Using the reviews with the replaced words and a list of the four issues, I calculated the relevant cosine similarities and picked the top 200 most similar reviews.

In [17]:
```
from sklearn.metrics.pairwise import cosine_similarity
from nltk.tokenize import word_tokenize
from math import log
import numpy as np
from collections import namedtuple
# from nltk.sentiment.vader import SentimentIntensityAnalyzer
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import csv
import re

Review = namedtuple('Review', 'restaurant review_text rating')
REVIEWS_FILE_PATH = 'ja3.csv'
RATING_REGEX_PATTERN = 'alt=\"([0-5])'
MINIMUM_RESTAURANT_RATED_REVIEWS_COUNT = 10
MINIMUM_RESTAURANT_SENTIMENT_REVIEWS_COUNT = 8


def get_attributes_idf_values(attributes, reviews_texts):
    attributes_freqs = [0] * len(attributes)

    for review in reviews_texts:
        review_tokens = word_tokenize(review.lower())
        for i in range(len(attributes)):
            if attributes[i] in review_tokens:
                attributes_freqs[i] += float(1 / len(reviews_texts))

    attributes_idfs = list(map(lambda attribute_freq: log(1 / attribute_fr
    return attributes_idfs


def get_tfidf_values(reviews, attributes, idf_values):
    tfidf_values = np.zeros((len(reviews), len(idf_values)))

    for i in range(len(reviews)):
        review_tokens = word_tokenize(reviews[i])
        for j in range(len(attributes)):
            attribute_count = review_tokens.count(attributes[j])
            tfidf_values[i][j] = attribute_count * idf_values[j]
    return tfidf_values


```

```
def get_top_reviews_by_cosine_similarity(attributes, reviews):
    reviews_similarities = []

    reviews_texts = [review.review_text for review in reviews]

    idf_values = get_attributes_idf_values(attributes, reviews_texts)
    reviews_tfidfs = get_tfidf_values(reviews_texts, attributes, idf_value

    for i in range(len(reviews)):
        review_similarity = \
        cosine_similarity(reviews_tfidfs[i].reshape(1, -1), np.asarray(idf
        reviews_similarities.append((review_similarity, reviews[i]))

    sorted_reviews_similarities = sorted(reviews_similarities, key=lambda

    return sorted_reviews_similarities[:200]

def import_reviews():
    reviews = []
    with open(REVIEWS_FILE_PATH, 'r', encoding="utf8") as csv_file:
        csv_reader = csv.reader(csv_file, quotechar='"', delimiter=',', qu
        for row in csv_reader:
            restaurant = row[2]
            review_text = row[4]
            rating = extract_rating(row[5])
            reviews.append(Review(restaurant=restaurant, review_text=revie
    return reviews

def extract_rating(html_code):
    if html_code == '':
        return None
    rating_search = re.search(RATING_REGEX_PATTERN, html_code, re.IGNORECA

    if rating_search:
        return int(rating_search.group(1))

    return None

```

```
/anaconda3/lib/python3.6/site-packages/nltk/twitter/__init__.py:20: UserWarning: The twython library has not been installed. Some functionality from the twitter package will not be available.
  warnings.warn("The twython library has not been installed. "
```

In [18]:

```
reviews = import_reviews()
reviews_texts = [review.review_text for review in reviews]
reviews_filtered = list(filter(lambda review: review.review_text is
attributes = ['service', 'food', 'price', 'location']

```

Having identfied the 200 reviews with the highest cosine similarity, I filtered them out from the list of reviews.

In [19]:
```
# creating a list of the 200 reviews with the highest cosine similar
top_reviews_by_cosine_similarity = get_top_reviews_by_cosine_similar
```

In [20]:
```
top_reviews_by_cosine_similarity
```

Out[20]:
```
[(1.0,
  Review(restaurant='Haiku Japanese Restaurant', review_text="I've bee
n to this location a handful of times. \xa0I have received exceptional
service during each visit. \xa0The servers are very friendly. \xa0The
hot tea with lavender was AMAZING! The lunch special is hard to beat-
a lot of food for a reasonable price. \xa0My brother had the Chirashi
lunch plate and he loved it. \xa0I usually stick to sushi rolls. \xa0N
ever been disappointed.", rating=None)),
 (1.0,
  Review(restaurant='D K Sushi Restaurant', review_text='Nice combinat
ion of different Asian dishes including Korean and Japanese food. We w
ere pleasantly surprised at the nice atmosphere and great service comp
ared to the interesting location. Our waitress was friendly and helpfu
l regarding the menu. We went there for sushi, but ended up eating Kor
ean, which was quite tasty. The bulgogi (barbequed marinated beef) was
a bit dry, but still finger-licking good. The japchae (stir fried nood
les) was awesome. The price was very affordable, enough to make us reg
ulars!', rating=None)),
 (1.0,
```

In [21]:

```python
#Display cosine similarity
top_rest_cos=[]
for i in range(0,len(top_reviews_by_cosine_similarity)):
    top_rest_cos.append((top_reviews_by_cosine_similarity[i][1][0],t
print(top_rest_cos)
```

```
[('Haiku Japanese Restaurant', 1.0), ('D K Sushi Restaurant', 1.0), ('
Drunk Fish Restaurant', 1.0), ('Sushi Ocean', 1.0), ('Nagoya Steak and
Sushi', 1.0), ('Cho Sushi Japanese Fusion', 0.9825387406792225), ('Yan
agi', 0.9825387406792225), ('Sushi Junai 2', 0.9795633040174123), ('Su
shi Junai 2', 0.9795633040174123), ('Midori Sushi', 0.9795633040174123
), ('D K Sushi Restaurant', 0.9795633040174123), ('Musashino Sushi Dok
oro', 0.9795633040174123), ('Umiya', 0.9795633040174123), ('D K Sushi
Restaurant', 0.9795633040174123), ('Izumi Japanese Sushi & Grill', 0.9
795633040174123), ('Beluga Japanese Restaurant', 0.9706937198712429),
('Uchi', 0.9698952137908505), ('Musashino Sushi Dokoro', 0.96090337915
42639), ('Shogun', 0.9607104711380637), ('Sa-Ten', 0.9607104711380637)
, ('Uchi', 0.9607104711380637), ('Soto Restaurant', 0.9607104711380637
), ('Sushi Junai 2', 0.9607104711380637), ('Ni-Kome Sushi + Ramen', 0.
9607104711380637), ('Soto Restaurant', 0.9607104711380637), ('Kemuri T
atsu-ya', 0.9607104711380637), ('Thai, How Are You?', 0.96071047113806
37), ('Sa-Tén - Canopy', 0.9607104711380637), ('Haru Sushi - formerly
known Hanabi', 0.9541807680415116), ('Sushi Junai 2', 0.95418076804151
16), ('D K Sushi Restaurant', 0.951085138502513), ('Haru Sushi - forme
rly known Hanabi', 0.951085138502513), ('Umiya', 0.9428579690183071),
```

In [22]:
```
top_cos_dict=dict((x, y) for x, y in top_rest_cos)
print(top_cos_dict)
```

```
{'Haiku Japanese Restaurant': 0.782616100252l016, 'D K Sushi Restauran
t': 0.951085138502513, 'Drunk Fish Restaurant': 0.9394194355727531, 'S
ushi Ocean': 0.8080493139631381, 'Nagoya Steak and Sushi': 0.808049313
9631381, 'Cho Sushi Japanese Fusion': 0.7826161002521016, 'Yanagi': 0.
7826161002521016, 'Sushi Junai 2': 0.7797081255743166, 'Midori Sushi':
0.9795633040174123, 'Musashino Sushi Dokoro': 0.7826161002521016, 'Umi
ya': 0.7826161002521016, 'Izumi Japanese Sushi & Grill': 0.80804931396
31381, 'Beluga Japanese Restaurant': 0.9706937198712429, 'Uchi': 0.960
7104711380637, 'Shogun': 0.8080493139631381, 'Sa-Ten': 0.8080493139631
381, 'Soto Restaurant': 0.7826161002521016, 'Ni-Kome Sushi + Ramen': 0
.7906316425888584, 'Kemuri Tatsu-ya': 0.8080493139631381, 'Thai, How A
re You?': 0.7879992312333786, 'Sa-Tén - Canopy': 0.7879992312333786, '
Haru Sushi - formerly known Hanabi': 0.7826161002521016, 'Uchiko': 0.7
897854977354688, 'Lavaca Teppan': 0.7826161002521016, 'Don Japanese Ki
tchen': 0.7826161002521016, 'Kanji Ramen': 0.886513575974612, 'Sushi Z
ushi': 0.7826161002521016, 'Nanami Sushi Bar & Grill': 0.7826161002521
016, 'Tokyo Steak House and Sushi Bar': 0.9394194355727531, 'JINYA Ram
en Bar': 0.7879992312333786, 'Haru Ramen & Yakitori': 0.88651357597461
2, 'Ramen Tatsu-Ya': 0.7826161002521016, 'Sushi Fever': 0.782616100252
1016, 'Kome Sushi Kitchen': 0.7826161002521016, 'Lucky Robot': 0.78261
61002521016, 'Umiya Leander': 0.7826161002521016, 'Raku Sushi and Asia
n Bistro': 0.886513575974612, 'Kura Revolving Sushi Bar': 0.7826161002
521016, 'Poke Me Long Time': 0.8080493139631381, 'Daruma Ramen': 0.782
6161002521016, 'Kai Sushi': 0.7826161002521016, 'Maiko': 0.78261610025
21016, 'Zen Japanese Food Fast': 0.7879992312333786, 'Tomodachi Sushi'
: 0.8080493139631381, 'BarChi Sushi': 0.7826161002521016, 'Soto - Sout
h Lamar': 0.7826161002521016, 'Sakura Sushi & Bar': 0.7826161002521016
, 'Bon Japanese Cuisine': 0.8080493139631381, 'Shogun Japanese Grill &
Sushi Bar': 0.7916633945544274, 'Sushi Hara': 0.7906316425888584, 'Sus
hi Junai': 0.7826161002521016, 'Ichiban': 0.7879992312333786, 'Michi R
amen': 0.7763758817843818, 'Teriyaki Madness': 0.7844894159618636, 'Ma
ki Toki': 0.7826161002521016, 'Fukumoto Sushi & Yakitori': 0.782616100
2521016, 'Sushi Nini': 0.7826161002521016, 'KOBE Japanese Steakhouse':
0.7826161002521016, 'Mikado Ryotei': 0.7826161002521016}
```

Task D. Perform sentiment analysis

Having this list of 200 reviews with the highest cosine similarities, I performed sentiment analysis and sorted them from high to low. In this project, using the VADER library, the sentiment score of each review was calculated as the sum of the sentiment of every word (using the polarity lexicon that is incorporated in the livrary) as well as the way that it every word is written (e.g. caps, exclamasion marks, etc.).

In [23]:
```
_restaurants_avg_sentiment_scores(reviews):
staurants_reviews_counts = {}
staurants_sentiments_sums = {}
staurants_avg_sentiment_scores = {}
alyser = SentimentIntensityAnalyzer()

 review in reviews:
 review_sentiment = analyser.polarity_scores(review.review_text)['compour
 restaurants_sentiments_sums[review.restaurant] = restaurants_sentiments_
 restaurants_reviews_counts[review.restaurant] = restaurants_reviews_cour

 restaurant in restaurants_reviews_counts.keys():
 if restaurants_reviews_counts[restaurant] < MINIMUM_RESTAURANT_SENTIMENT
    continue
 restaurants_avg_sentiment_scores[restaurant] = round(restaurants_sentime

urn sorted(restaurants_avg_sentiment_scores.items(), key=lambda restaurar

```

In [24]:
```
#Performing sentiment analysis and taking the average sentiment scor
top_rest_sent=(get_restaurants_avg_sentiment_scores([review_cosine_t
#print(top_rest_sent)
top_rest_sent_dict=dict((x, y) for x, y in top_rest_sent)
print(top_rest_sent_dict)

```

```
{'Haru Sushi - formerly known Hanabi': 0.98, 'Sushi Junai 2': 0.95, 'R
amen Tatsu-Ya': 0.77, 'Musashino Sushi Dokoro': 0.43, 'Sushi Zushi': 0
.38}
```

Task E. Restaurant recommendations based on cosine similarity and sentiment analysis

Based on tasks C and D, I made 3 restaurant recommendations. Note that in task D, multiple reviews may refer to the same restaurant, so I Used the average sentiment score for each restaurant. The three selected restaurants are the ones with the most positive sentiment and the highest cosine similarity to the four issues.

In [25]:

```
list_top_rest=[]
list_cosine=[]
for i in range(0,len(top_rest_sent)):
    list_top_rest.append(top_rest_sent[i][0])
    list_cosine.append((top_rest_sent[i][0],top_cos_dict[top_rest_se
top_rest_cos_dict=dict((x, y) for x, y in list_cosine)
print(top_rest_cos_dict)

```

```
{'Haru Sushi - formerly known Hanabi': 0.7826161002521016, 'Sushi Juna
i 2': 0.7797081255743166, 'Ramen Tatsu-Ya': 0.7826161002521016, 'Musas
hino Sushi Dokoro': 0.7826161002521016, 'Sushi Zushi': 0.7826161002521
016}
```

In [26]:

```
list_top_rest
```

Out[26]:

```
['Haru Sushi - formerly known Hanabi',
 'Sushi Junai 2',
 'Ramen Tatsu-Ya',
 'Musashino Sushi Dokoro',
 'Sushi Zushi']
```

Based on the above cosine similarity and sentiment analysis, the top 3 recommended restaurants along with their sentiment score and cosine similarity were the below:

1. Haru Sushi - formerly known Hanabi
2. Sushi Junai 2
3. Ramen Tatsu-Ya

In [27]:
```
rest_table = pd.DataFrame(index=list_top_rest, columns=('Cosine Simi
for row in rest_table.index:
    rest_table.loc[row]['Cosine Similarity']=top_rest_cos_dict[row]
    rest_table.loc[row]['Avg.Sentiment Score']=top_rest_sent_dict[ro
print(rest_table)
```

```
                                   Cosine Similarity Avg.Sentiment Sco
re
Haru Sushi - formerly known Hanabi          0.782616                0.
98
Sushi Junai 2                               0.779708                0.
95
Ramen Tatsu-Ya                              0.782616                0.
77
Musashino Sushi Dokoro                      0.782616                0.
43
Sushi Zushi                                 0.782616                0.
38
```

Task F. Recommendations based on average ratings

In order to check the significance and efficincy of my techniques, I made more restaurant reccommendations, using only the ratings (current practice of the website), calculating the three restaurants with the highest average rating.

In [28]:

```
def get_restaurants_average_ratings(reviews):
    rating_sums = {}
    rating_counts = {}
    rating_avgs = {}
    for review in reviews:
        if review.rating is None:
            continue
        rating_sums[review.restaurant] = rating_sums.get(review.rest
        rating_counts[review.restaurant] = rating_counts.get(review.
    for restaurant in rating_counts.keys():
        if rating_counts[restaurant] < MINIMUM_RESTAURANT_RATED_REVI
            continue
        rating_avgs[restaurant] = round(rating_sums[restaurant] / ra

    return rating_avgs


def get_top_restaurants_by_ratings(reviews):
    rating_avgs = get_restaurants_average_ratings(reviews)
    return sorted(list(rating_avgs.items()), key=lambda review: revi


```

In [29]:

```
# selecting the rating and restaurant name from every review droppin
rest_rating = yelp_data.loc[:,{'restaurant','rating'}].dropna().rese
rest_rating.head()
```

Out[29]:

	rating	restaurant
0	<img class="offscreen" height="303" src="https...	Misusushi
1	<img class="offscreen" height="303" src="https...	Drunk Fish Restaurant
2	<img class="offscreen" height="303" src="https...	Fukumoto Sushi & Yakitori
3	<img class="offscreen" height="303" src="https...	Sushi Zushi
4	<img class="offscreen" height="303" src="https...	Umiya

In order to use the rating data, I cleaned the "rating" column containing an html link that included the star rating as well, keeping only the value given by the reviewer.

In [30]:

```
# clearing out ratings html to keep only the value
stars = [int(*re.findall("(\d+).0 star rating", i)) for i in rest_ra

# replacing same in the restaurant - rating table
rest_rating['rating'] = pd.DataFrame(stars)
```

Then using the provided data from all the 12k reviews, I calculated the average rating for every restaurant and in order to have relevant reliable data, I filtered out restaurants with less than 10 reviews.

In [31]:

```
# identifying restaurants with less than k = 10 reviews
k = 10
no_review_restaurants = []

for i in range(len(rest_rating['restaurant'].value_counts())):
    restaurant = rest_rating['restaurant'].value_counts().index.toli
    reviews = rest_rating['restaurant'].value_counts()[i]
    if reviews < k:
        no_review_restaurants.append(restaurant)
no_review_restaurants
```

Out[31]:

```
['K-Bow Tie',
 'Momo Sushi',
 'Express Teriyaki & Grill',
 'Miyako Yakitori & Sushi',
 'Little Tokyo']
```

In [32]:

```
# filtering out the restaurants with less than k reviews
rest_rating_10 = rest_rating[-rest_rating['restaurant'].isin(no_revi
```

In [33]:

```
# calculating average rating for each restaurant and sorting them in
most_stars = rest_rating_10.groupby(['restaurant'], as_index=False).
most_stars
```

80	Zen Japanese Food Fast	3.360360
79	Yoshi Ramen	3.343434
31	Maiko	3.317204
67	Sushi Zushi	3.267196
70	Thai, How Are You?	3.211268
42	Ni-Kome Sushi + Ramen	3.181818
71	Tokyo Steak House and Sushi Bar	3.177083
32	Maki Toki	3.176136
10	Drunk Fish Restaurant	2.921053
54	Shogun Japanese Grill & Sushi Bar	2.881443
20	Japan Cafe	2.739130

81 rows × 2 columns

Based on the rating scores alone, it turned out that the top three restaurants were:

1. Baja St Tacos & Coastal Cuisine
2. Dawa Sushi
3. Otoko

These results were not really reliable and of smaller value related to the ones through the context analytics for many reasons, which I summarise below.

Conclusions

Based on the above results of both methods, I identified the following:

1. Restaurants which are highly rated in terms of the attributes on the basis of which the user is seeking recommendation, appears at a lower rank when listed solely based on ratings. Therefore, simply building the recommendation system on user ratings does not meet the requirements of the user looking for recommendations.
2. The restaurant 'Haru Sushi - formerly known Hanabi' is the most desirable restaurant in terms of the user mentioned attributes. However, it is ranked 9th in terms of ratings and far behind restaurants which are barely mentioned when the four attributes (Service, Food, Price,Location) are discussed in the reviews.
3. Based on the ratings, the best japanese restaurant is "Baja St Tacos & Coastal Cuisine" which is not a traditional japanese restaurant and while it is highly rated by people that like the combination of tex-mex and japanese cuzine, it is not highly correlated to japanese "food" and as a result it may not be suitable for people looking for japanese flavors.
4. The list of 200 reviews with the highest cosine similarity includes no review of the "top three based on ratings" restaurants, which indicates that while those restaurants get positive feedback same is not strongly correlated to the four identifed aspects and as a result they do not present a good value proposition.

As a result, I can say that the "top three based on ratings" recommended restaurants, did not meet the requirements of the users looking for recommendations. The website's recommendation system would be much improved incorporating aspects and features to the search engine (e.g. Service, Food, Price, Location) helping users find exactly what they are looking for.

In [ ]:

```

```