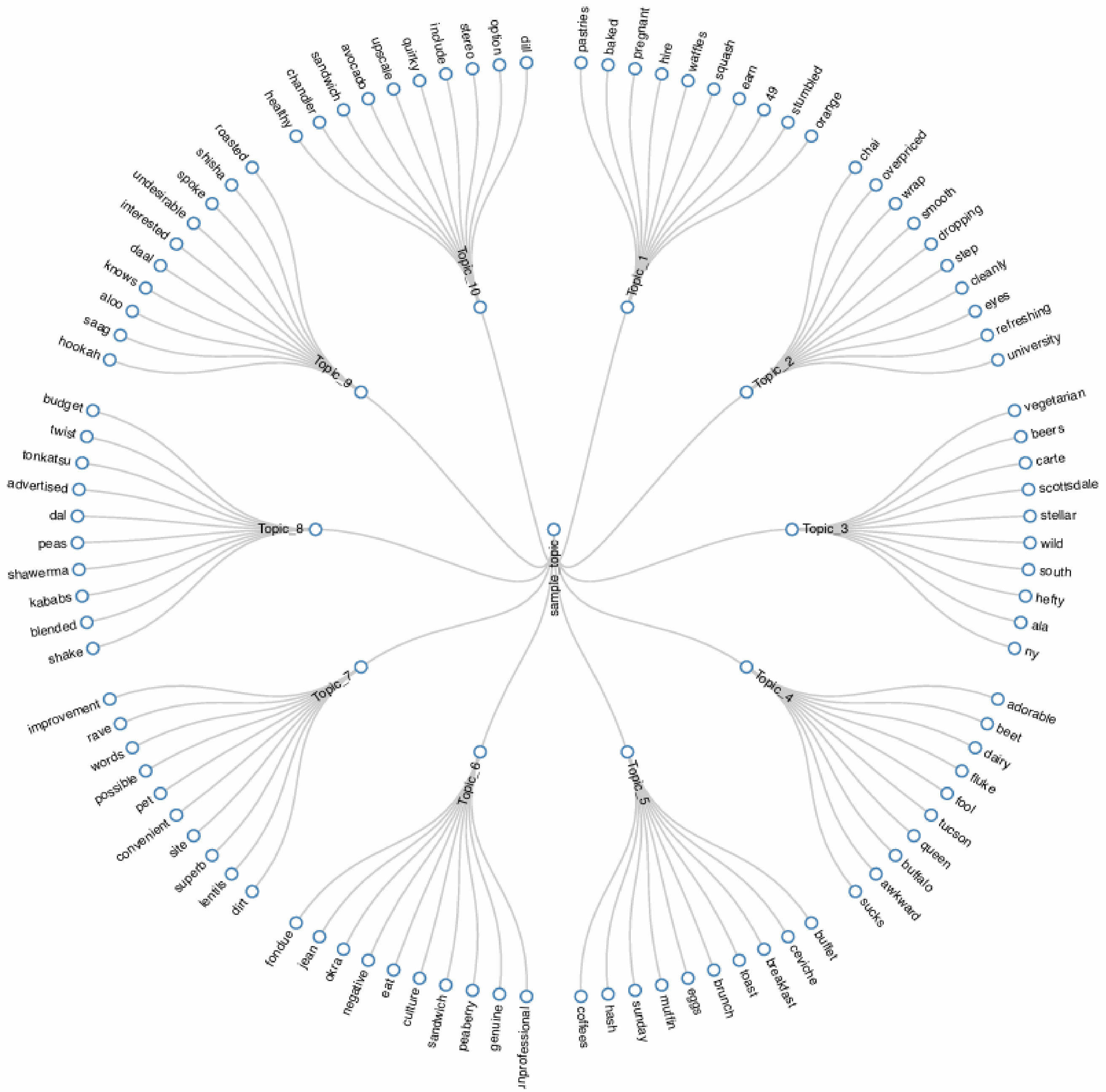
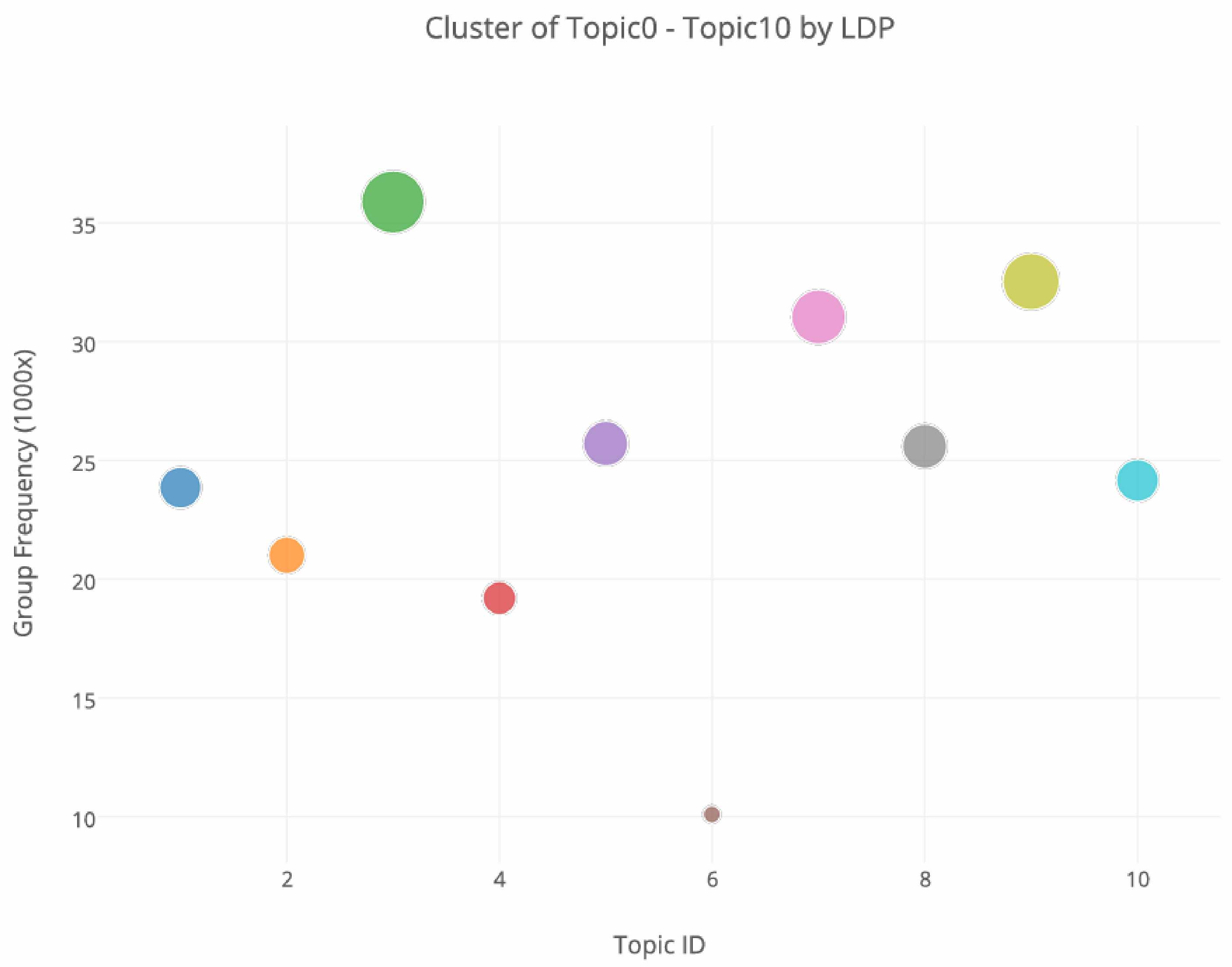
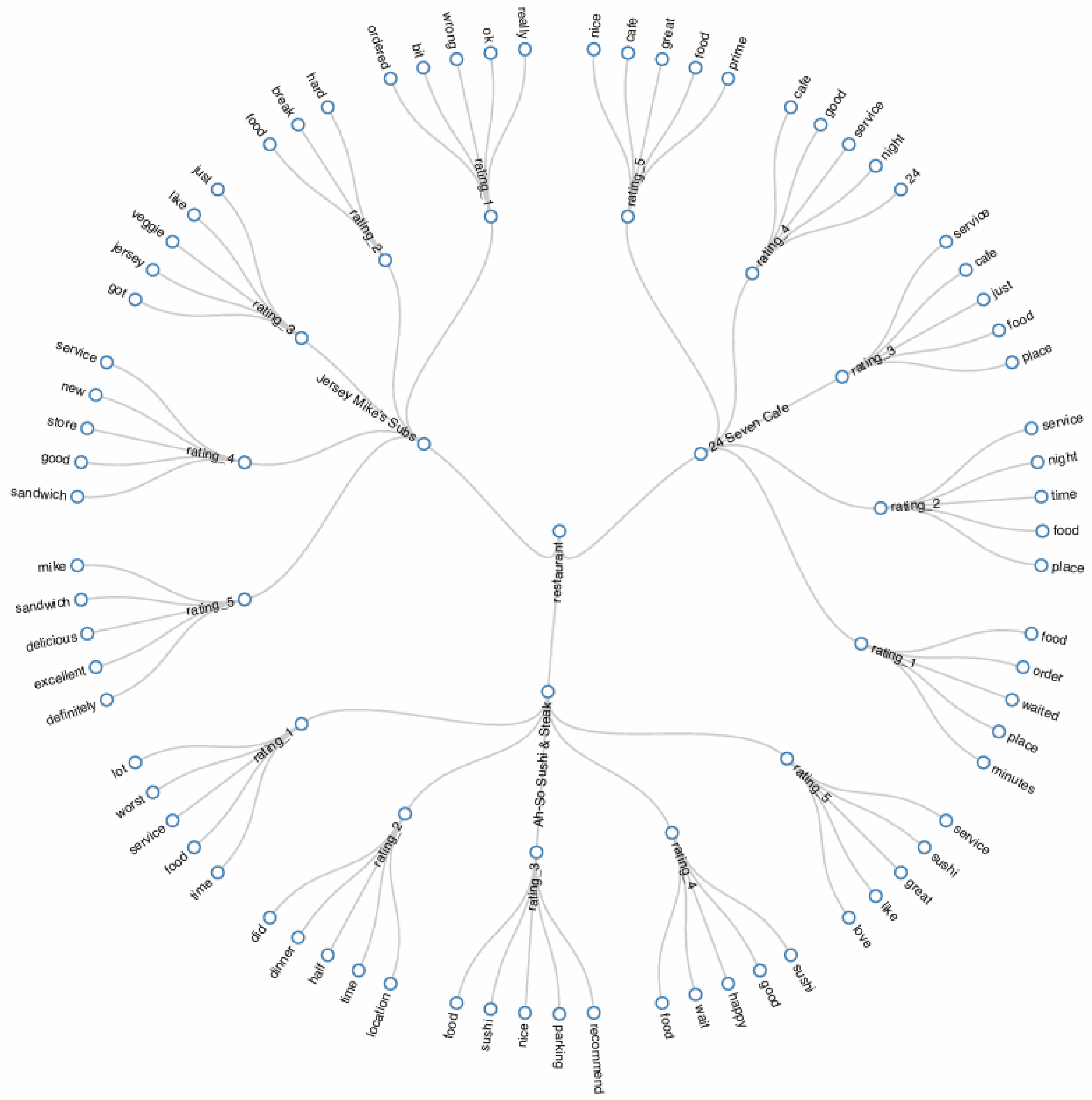
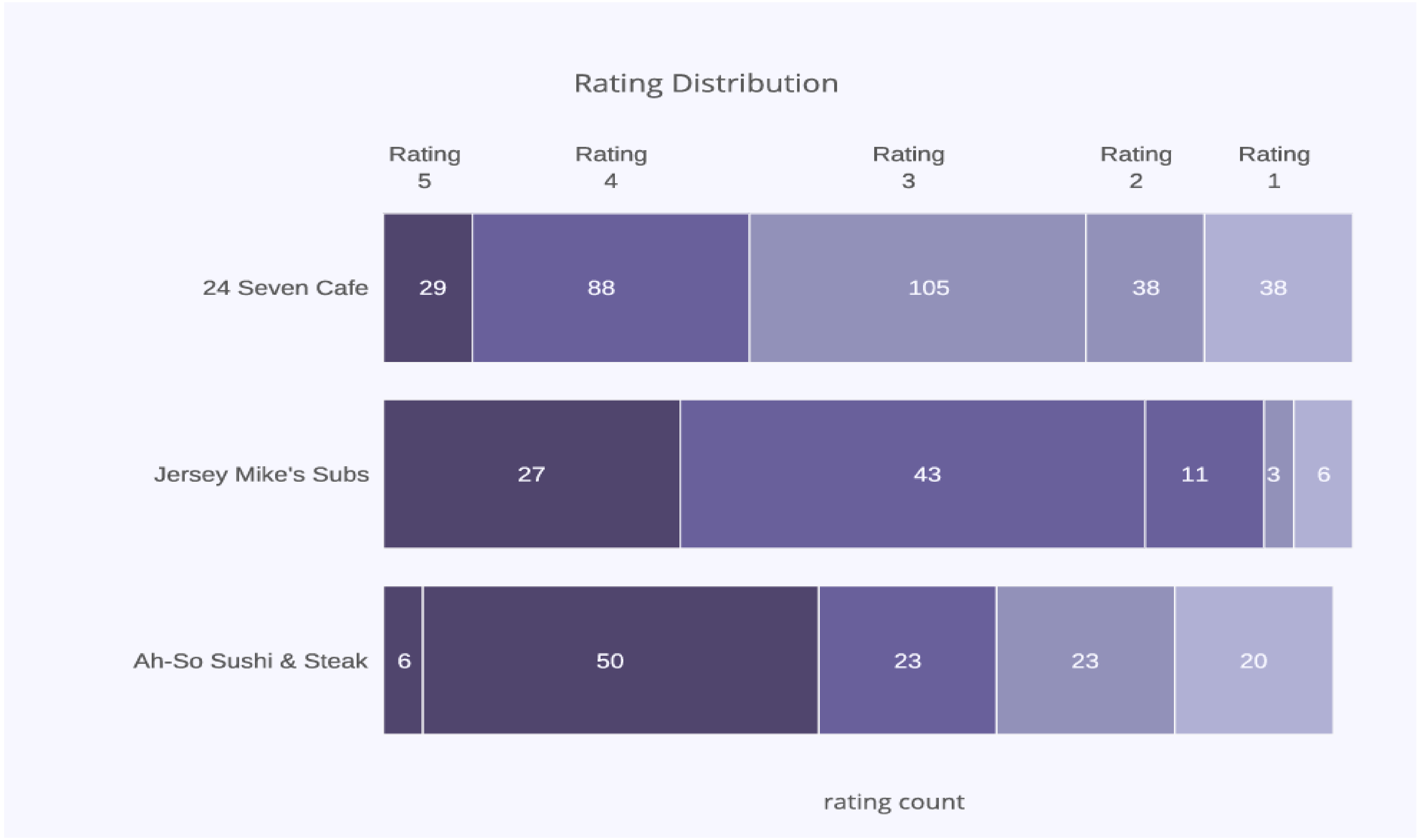

sample_topic
Topic_1
pastries
baked
pregnant
hire
waffles
squash
earn
49
stumbled
orange
Topic_2
chai
overpriced
wrap
smooth
dropping
step
cleanly
eyes
refreshing
university
Topic_3
vegetarian
beers
carte
scottsdale
stellar
wild
south
hefty
ala
ny
Topic_4
adorable
beet
dairy
fluke
fool
tucson
queen
buffalo
awkward
sucks
Topic_5
buffet
ceviche
breakfast
toast
brunch
eggs
muffin
sunday
hash
coffees
Topic_6
unprofessional
genuine
peaberry
sandwich
culture
eat
negative
okra
jean
fondue
Topic_7
dirt
lentils
superb
site
convenient
pet
possible
words
rave
improvement
Topic_8
shake
blended
kababs
shawerma
peas
dal
advertised
tonkatsu
twist
budget
Topic_9
hookah
saag
aloo
knows
daal
interested
undesirable
spoke
shisha
roasted
Topic_10
healthy
chandler
sandwich
avocado
upscale
quirky
include
stereo
option
dill


Cluster of Topic0 - Topic10 by LDP
Group Frequency (1000x)
35
30
25
20
15
10
2
4
6
8
10
Topic ID

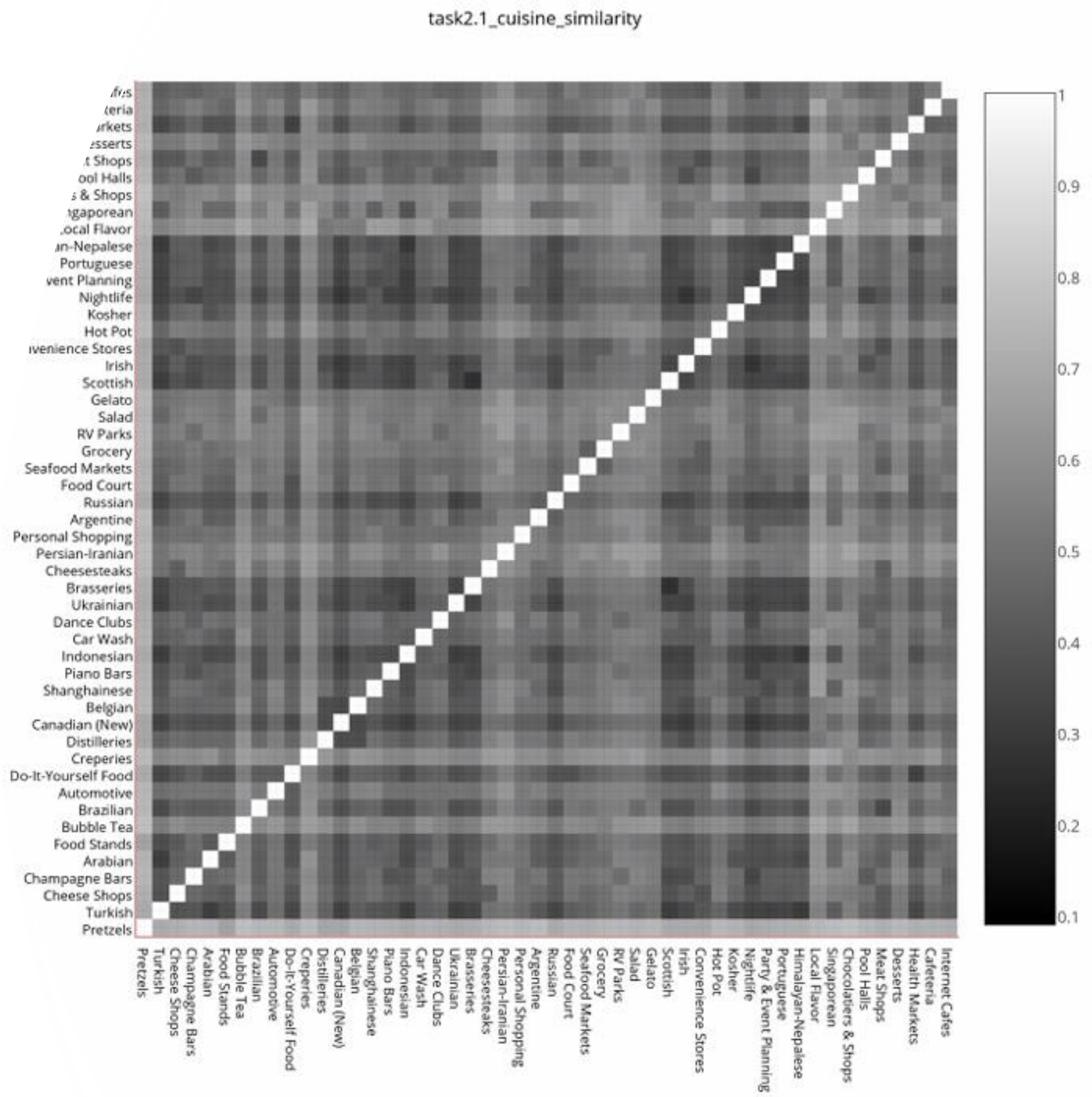

restaurant
Jersey Mike's Subs
rating_1
ordered
bit
wrong
ok
really
rating_2
food
break
hard
rating_3
just
like
veggie
jersey
got
rating_4
service
new
store
good
sandwich
rating_5
mike
sandwich
delicious
excellent
definitely
24 Seven Cafe
rating_5
nice
cafe
great
food
prime
rating_4
cafe
good
service
night
24
rating_3
service
cafe
just
food
place
rating_2
service
night
time
food
place
rating_1
food
order
waited
place
minutes
Ah-So Sushi & Steak
rating_1
lot
worst
service
food
time
rating_2
did
dinner
half
time
location
rating_3
food
sushi
nice
parking
recommend
rating_4
food
wait
happy
good
sushi
rating_5
service
sushi
great
like
love


Rating Distribution
Rating 5
Rating 4
Rating 3
Rating 2
Rating 1
24 Seven Cafe
29
88
105
38
38
Jersey Mike's Subs
27
43
11
3
6
Ah-So Sushi & Steak
6
50
23
23
20
rating count

Task 2.1: Cuisine Map VIsualization

For this task, I randomly selected 50 categories from the provided dataset and followed these steps:

1. Preprocessing each category representation:
   1. Tokenize the text using nltk.tokenize.
   2. Remove stop words using nltk.corpus.stopwords.
   3. Eliminated punctuation such as " , ! @ # $~
   4. Discard words with low frequency(occurences equal to 1).
   5. Apply stemming with nltk.stem.LancasterStemmer
2. **Word count calculation**: Compute the word coun for each preprocessed category representation.
3. **Cosine distance calculation**: Measure the cosine distance between each pair of category representation, without apply TF-IDF.



Task 2.2: Enhancing the Cuisine Map

For this task, I utilized Gensim and NLTK, following these steps:

1.**Text representation for each category:** Compile all review associated with a specific category as its text representation.

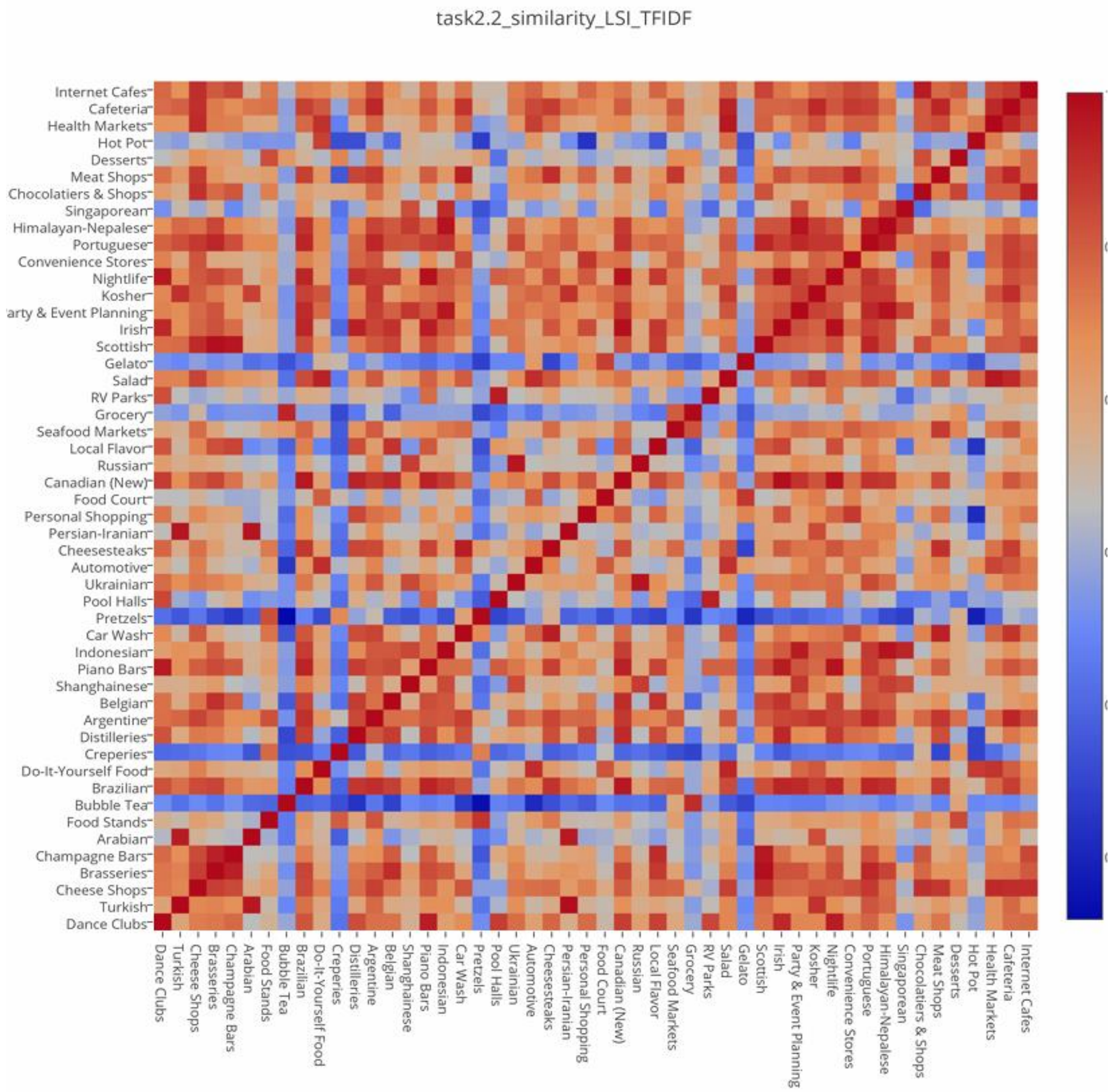
2. **Preprocessing:** Apply the same preprocessing steps outlined in Task 2.1.1 .

3. **Model training**: Train the preprocessed reviews using both LDA and LSI models:

   1. Process the category reviews using TF-IDF

   2. Train LDA/LSA models with 10 topics to generate the dictionary, matrixSimilarity index, and model instance.

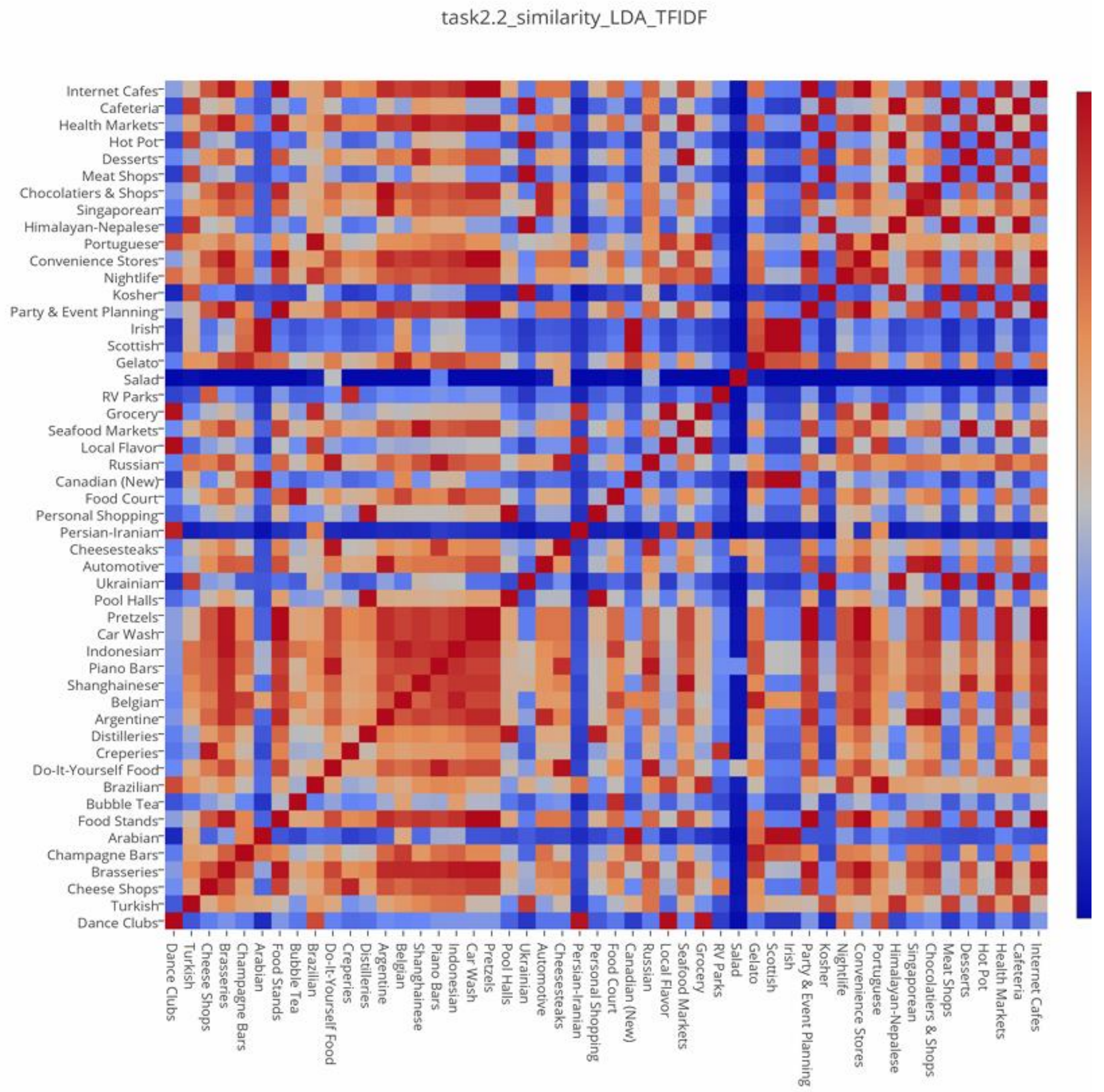
4. **Similarity calculation**: Determine the similarity between each pair of cuisines based on the results from the previous step.



From the graphs below, two trends are apparent:

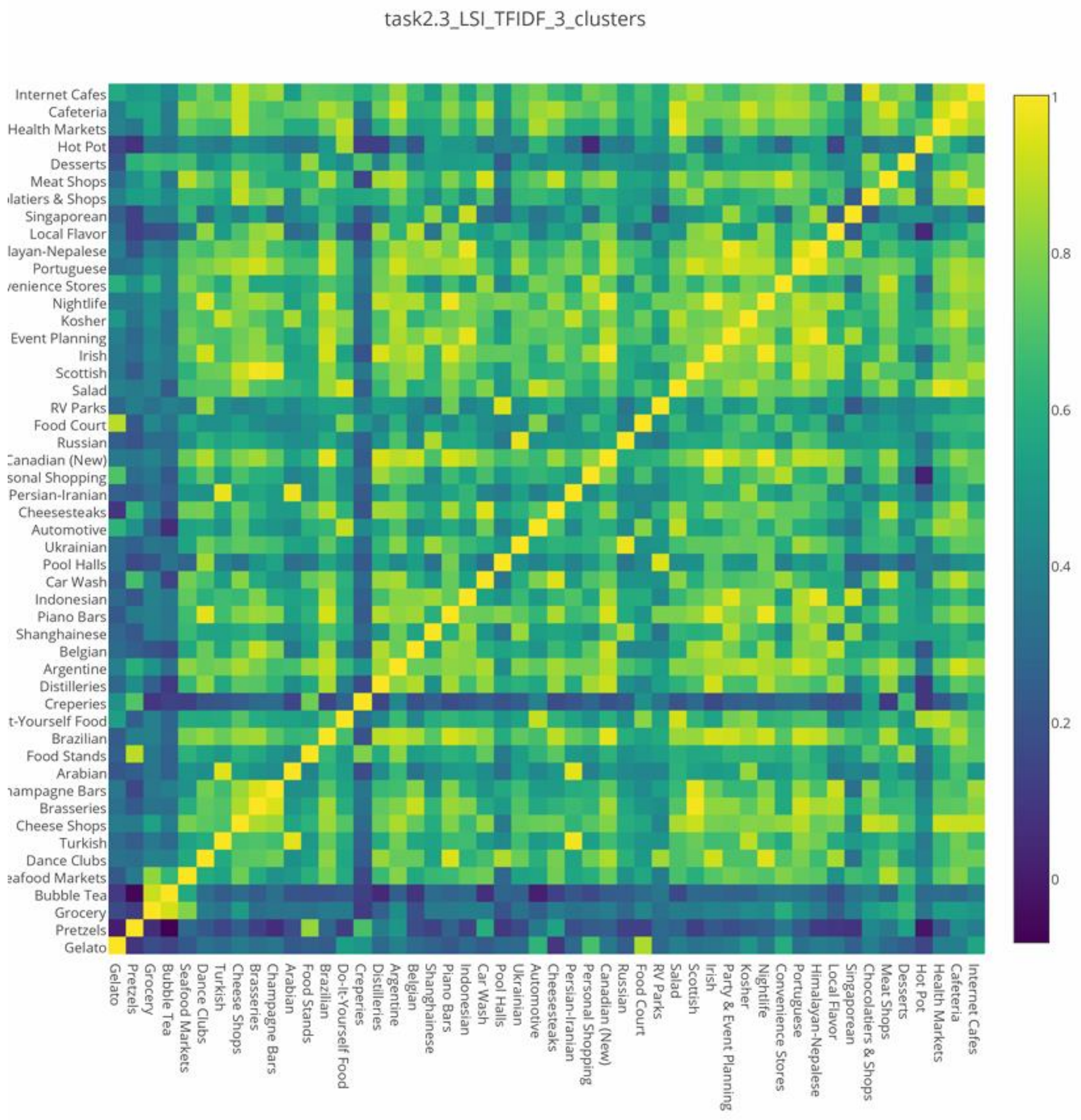
1. The similarity results from LSI model closely resemble the cosine distance similarity observed in Task 1, while the LDA model exhibits a noticeably different similarity pattern.

2. The LDA model tends to produce a more clustered grouping, where multiple cuisines appear similar, whereas the clustering in the LSI model is less distinct.

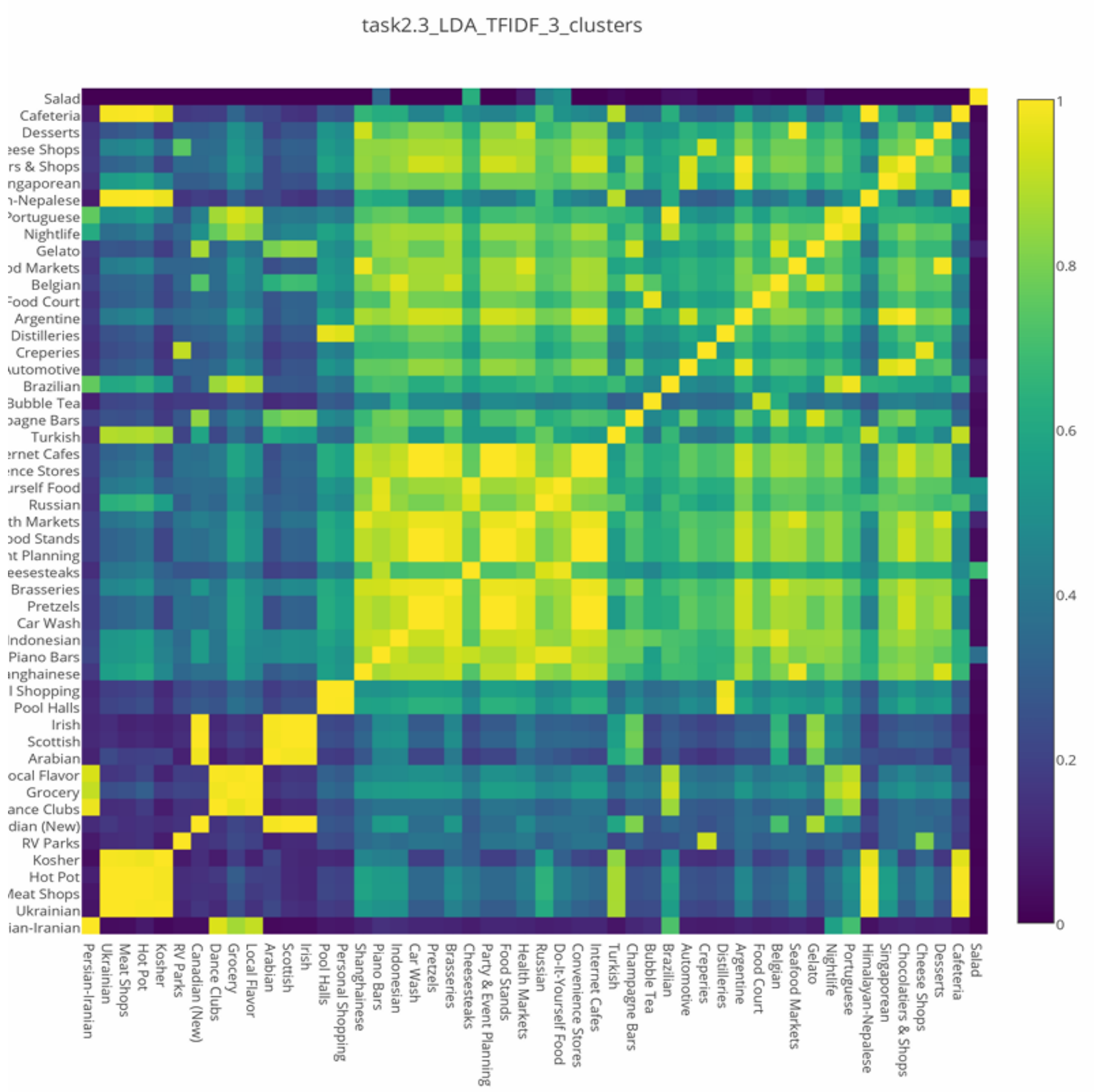


Task 2.3: Integrating Clustering into the Cuisine Map

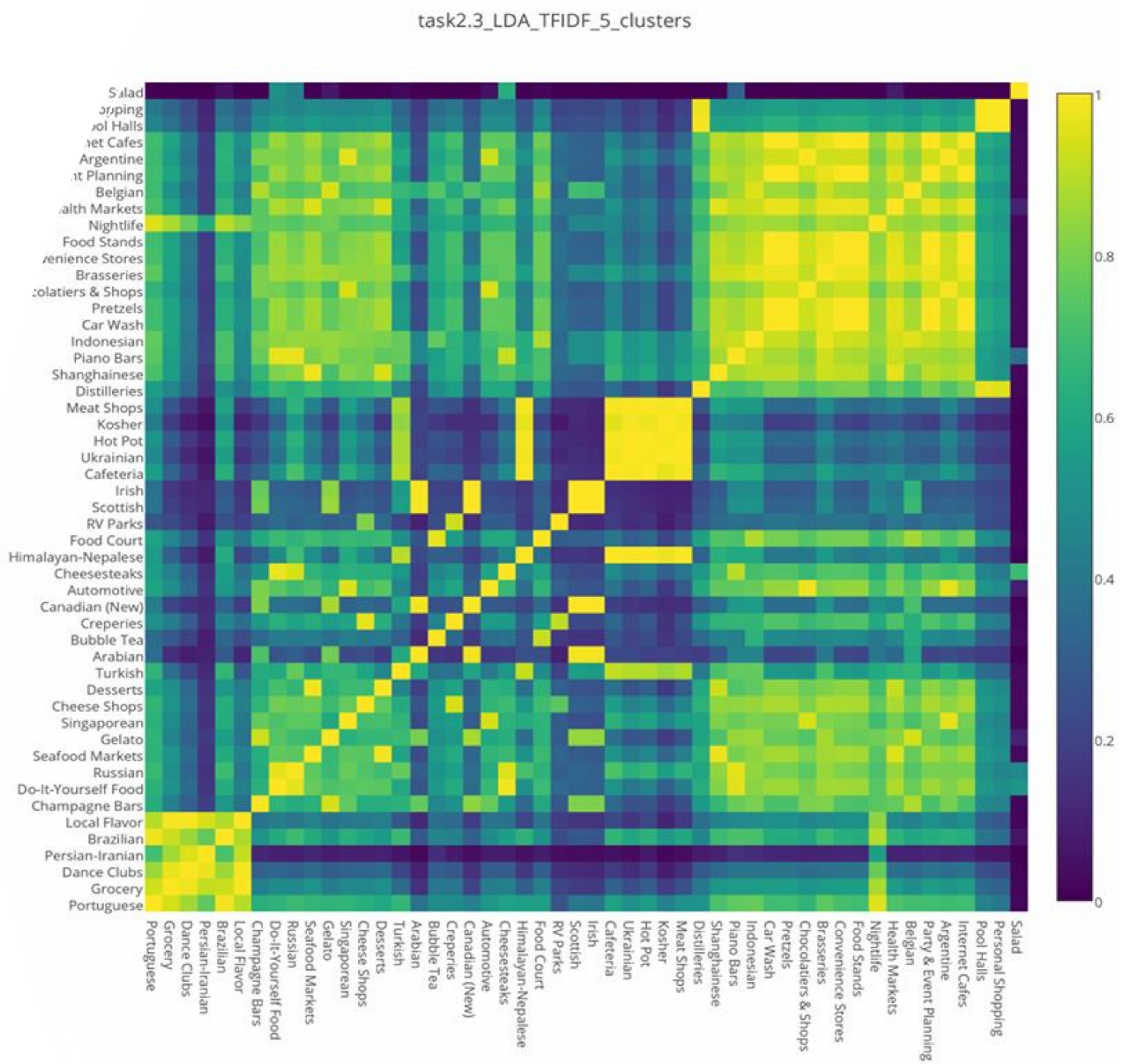
Using the same processing steps as in Task 2.2 for the text representation of categories, I leveraged the available similarity data between category pairs. Based on their mutual similarity, I organized the categories by grouping similar ones closer together.



I, then, tried to visualize cluster difference based on LDA model with different cluster



- From the above results we can see that with the same text representation, LDA provides a better result than LSI in terms of similarity clustering, the cluster is more identifable using LDA model.



Task 3 : Dish Recognition

Task 3.1: Manual Tagging

Objective

Refine the label list for a specific cuisine provided in the given file.

Tools Used: SegPhrase

Since there were no pre-existing labels available for training the input file, I used the default label information in SegPhrase. This allowed me to extract labels related to Chinese cuisine reviews and tips. Based on these labels, I inferred missing dishes.

Here are the steps I followed for this task:

- **Classifying Reviews and Tips**
  - Dish names appeared in both the review file and tips file.
  - I categorized the reviews and tips for all restaurants by cuisine (e.g., Chinese restaurants) and recorded them in a file named `Chinese-reviews.txt`.
- **Training with SegPhrase**
  - I trained the `Chinese-reviews.txt` file using SegPhrase with its default settings (default labels).
  - This generated a `wiki.label.auto` file with labels marked as 0 or 1.
- **Merging and Resolving Conflicts**
  - As the instructions noted, some provided labels might be incorrect. To address this:
    - I merged the labels from `wiki.label.auto` with the provided labels in the `task1_label` file to identify discrepancies.

    Steps Taken:

    A. Loop through all provided labels to identify conflicts:

    - If a label did not exist in the `wiki.label.auto` file, I skipped it to avoid errors, as this was a safer approach than switching values (e.g., from 0 to 1 or vice versa).
    - If a label existed in the `wiki.label.auto` file, I included it in the output file, adopting the corresponding value from the `wiki.label.auto` results.

B. Identifying Missing Labels:

1. I reviewed all labels in the `wiki.label.auto` results to check for missing ones.
2. Using manual filtering, I included only the labels directly related to dish phrases since the volume of labels was manageable.

Then Submit the final task1 label file to auto grader.

Other Possibilities

Initially, I attempted to use the wiki.label.auto file from SegPhrase directly, but the results were suboptimal as it included unrelated phrases.

Analysis

The `wiki.label.auto` file contained unusual phrases like "Los Angeles" and "South California." These phrases likely appeared in reviews because people mentioned them when discussing locations. While specific restaurant addresses could be useful, these phrases were not relevant to dish names and needed to be excluded.

Additionally, unrelated terms like "bra" were also included in the file. To ensure the final output was cohesive and focused solely on dish names, I manually removed such irrelevant items

Task 3.2 Mining Additional DIsh Names

Objective: Expand missing dish names

Tools Used: ToPMine, SegPhrase

What I did:

1.Train the review and tips of Chinese category using SegPhrase

2.Train the CHinese-reviews.txt file using ToPMine

3.Combine labels of first and second part to the task2_output file.

4. Finally, I removed duplicated dishes in task2_output file to get the result which contains around 6,900 items.

Dish and Resturant Ranking

The aim of Tasks 4 and 5 is to use identified dish names to assist people in making better dining choices. Task 4 focuses on four different methods for ranking the popularity of dishes within a specific cuisine list. It begins with a straightforward count-based ranking and progresses to a more nuanced sentiment-based approach. Task 5 delves into recommending restaurants for specific dishes by analyzing a relevant subset of reviews and applying a similar ranking methodology.The focus remains on Indian cuisine, utilizing the dish list derived from the correct annotations in Task 3 for Tasks 4 and Task 5. These dishes include:

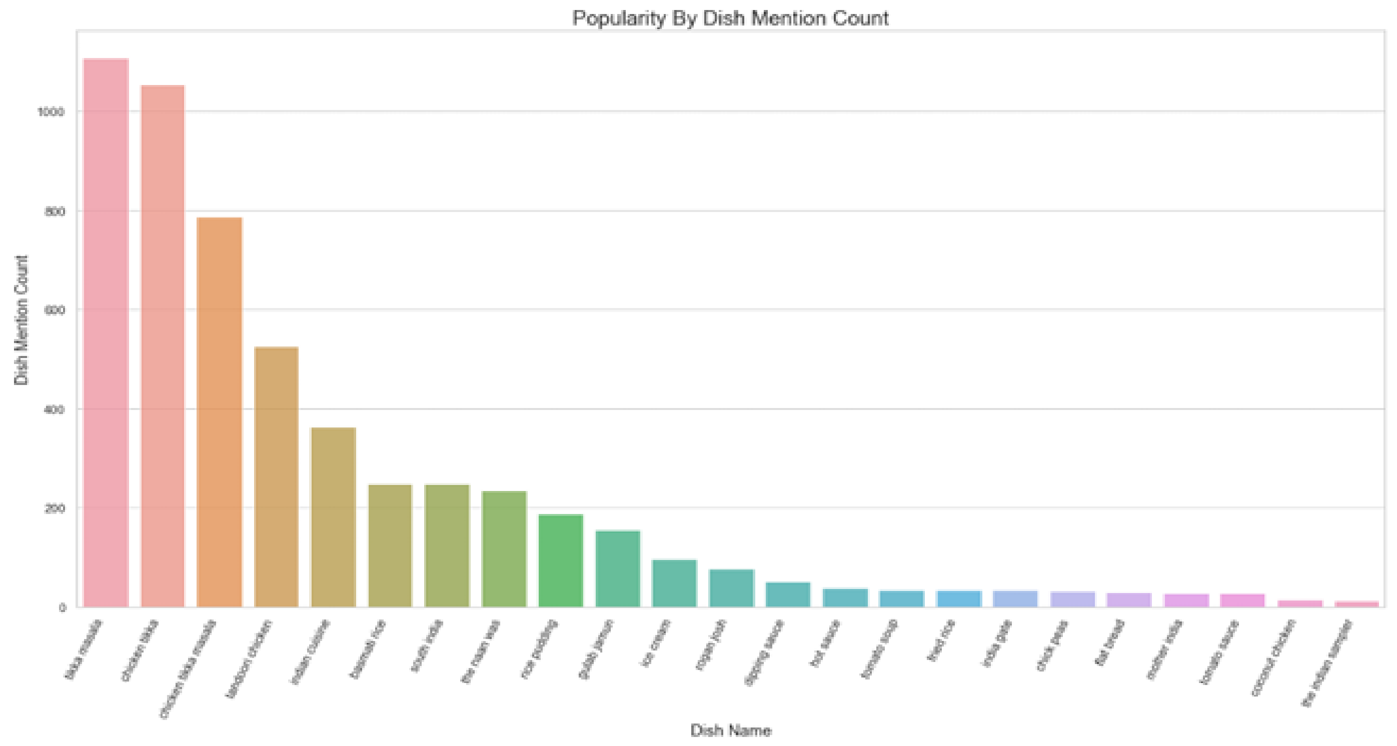
```
['chick peas',
 'chicken tikka',
 'flat bread',
 'tandoori chicken',
 'rogan josh',
 'mother india',
 'gulab jamun',
 'basmati rice',
 'rice pudding',
 'hot sauce',
 'fried rice',
 'ice cream',
 'south india',
 'tomato soup',
 'indian cuisine',
 'chicken tikka masala',
 'tomato sauce',
 'india gate',
 'the indian sampler',
 'dipping sauce',
 'the naan was',
 'tikka masala',
 'coconut chicken']
```

To facilitate the subsequent analysis, the reviews are filtered to include only those related to Indian restaurants. Additionally, the corresponding star ratings and restaurant names are extracted, as these will be essential for performing counts at the restaurant level or incorporating ratings into the rankings.

Task 4: Dish Ranking

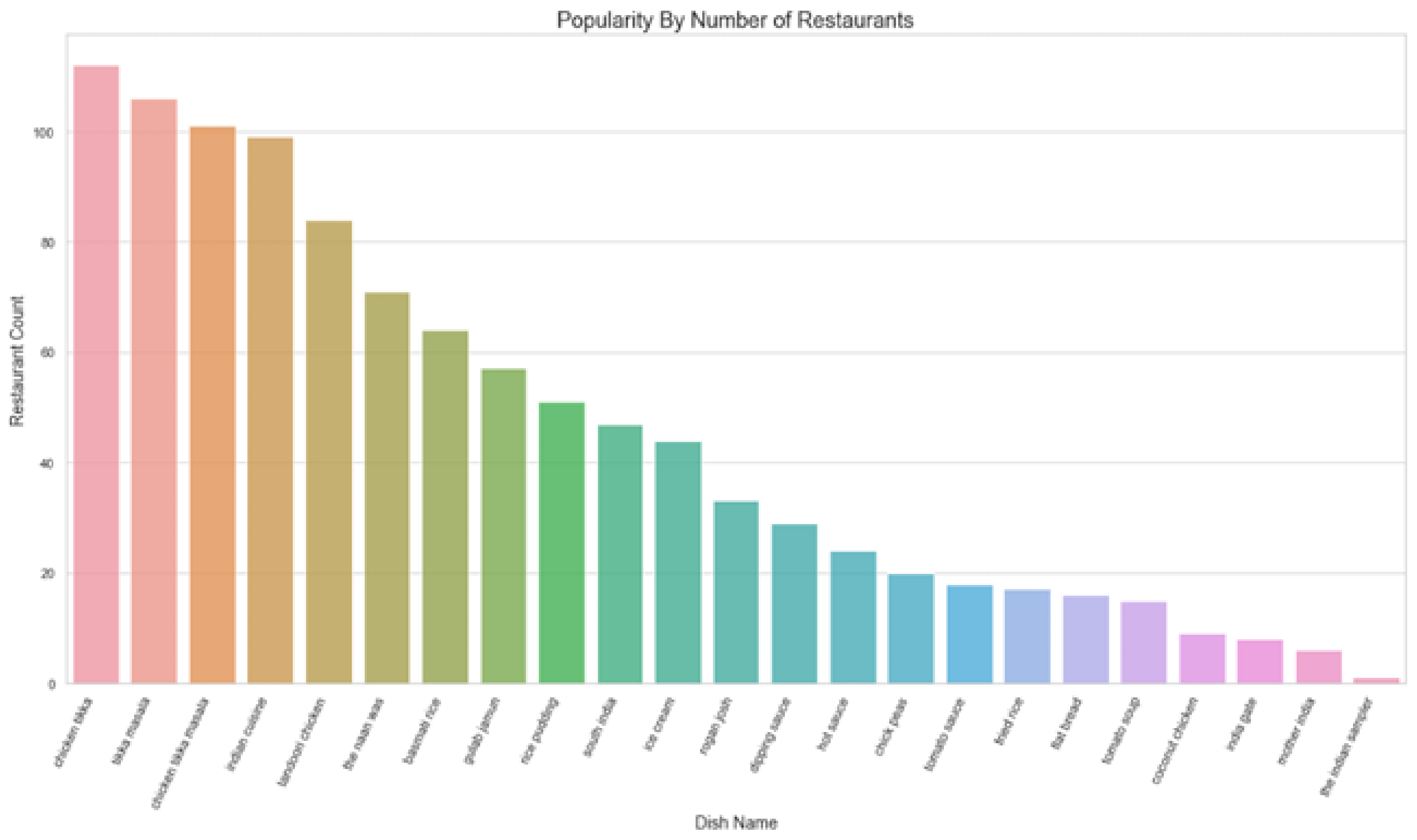
Task 4.1: Popularity By Dish Mention Count

The most straightforward method involves counting the frequency of a dish's mentions across all reviews of restaurants serving a specific cuisine. The initial ranking approach implements this method by calculating the number of mentions for each dish within the reviews of Indian cuisine. The dishes are then ranked based on these counts, with visualization achieved through a barplot generated using the Seaborn library.



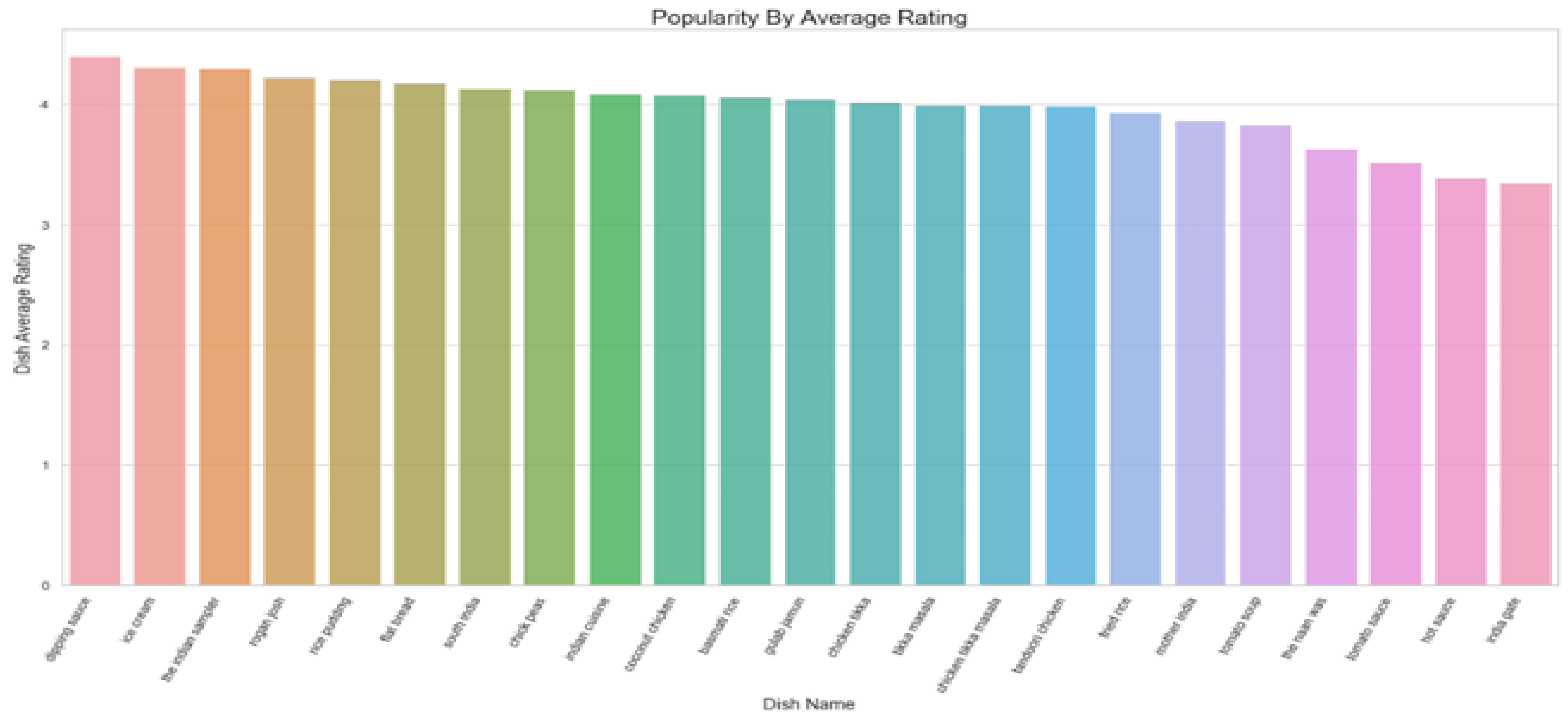
Task 4.2: Popularity by Number of Restaurants

This approach involves counting the total number of restaurants whose reviews mention a specific dish. This method helps mitigate the potential overemphasis caused by reviews that repeatedly reference the same dish. The Seaborn barplot is employed for visualization, as it effectively highlights the rankings of various dishes and facilitates comparisons across different ranking methods. This visualization technique will be consistently used throughout Tasks 4 and 5.



Task 4.3: Popularity By Average Rating

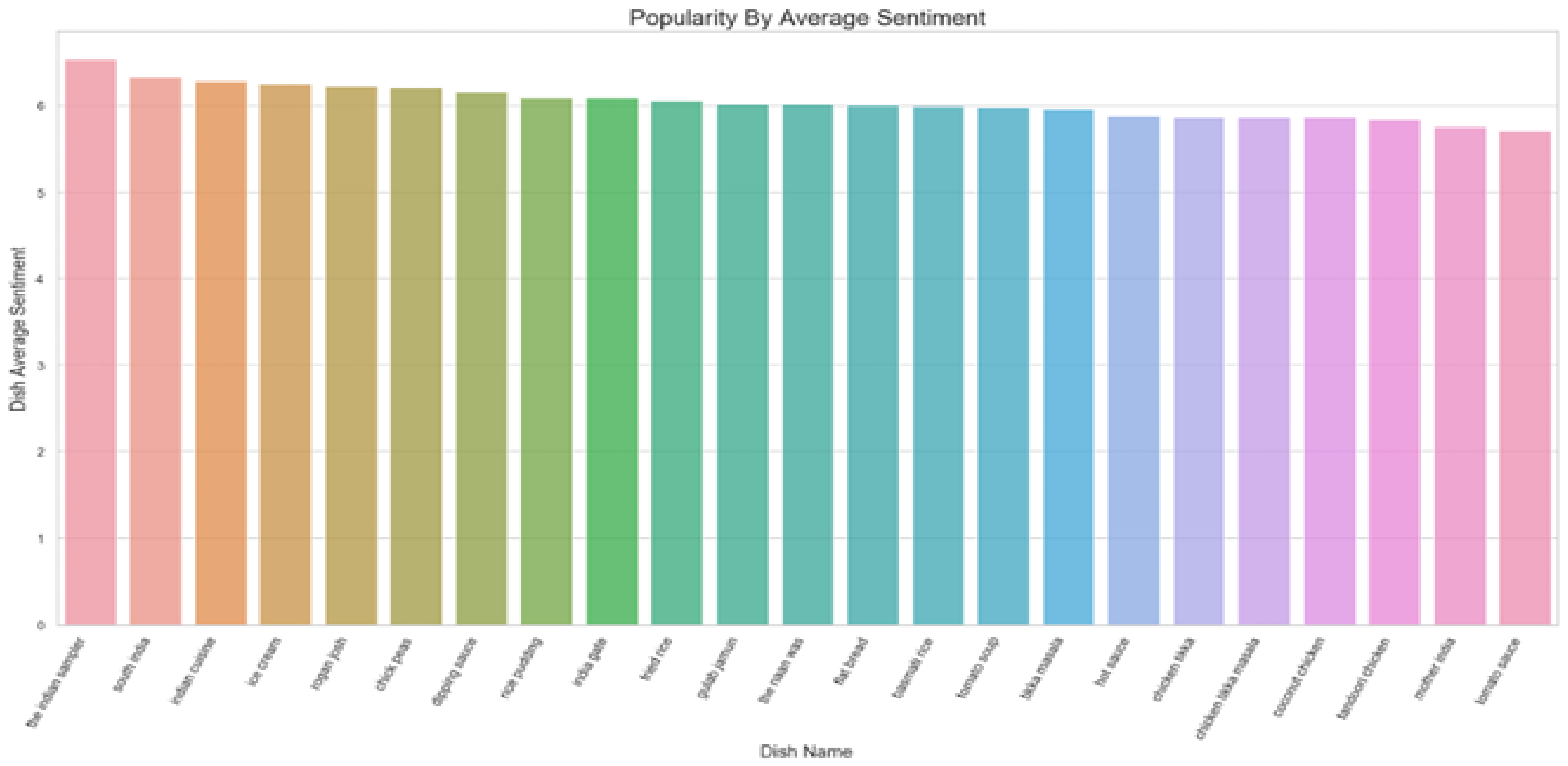
This method incorporates user feedback by leveraging star ratings to assess dish popularity. For each dish, an average ratingis calculated, with higher averages resulting in higher rankings. Reviews with a 3-star rating are excluded from the analysis, as they are generally neutral and provide limited insight regarding recommendation or non-recommendation.



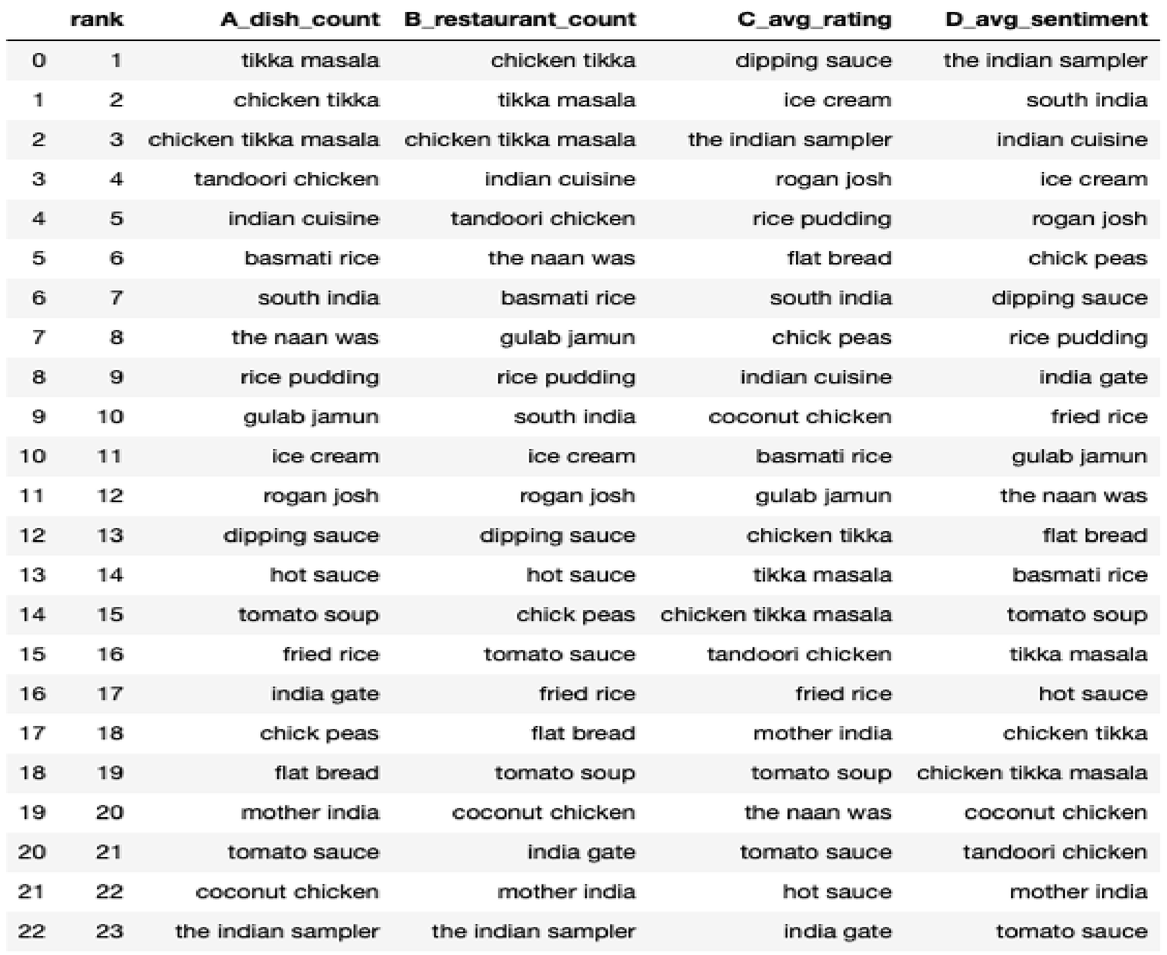
Task 4.4: Popularity By Sentiment

Finally, user sentiment is incorporated into the rankings by analyzing the average sentiment associated with each dish. This is achieved using TextBlob, a widely used Python library for sentiment analysis. TextBlob offers a convenient API that analyzes text and returns a tuple containing polarity and subjectivity. The focus is on polarity, which ranges from -1 to 1. For simplicity, the polarity values are transformed to a scale of 0 to 10 to facilitate averaging sentiment scores. The following formula was utilized for this transformation.

```
Formula to transform range from [a,b] to [c,d], where we have [-1,1] to
[0,10]
y = (x-a)(d-c)/(b-a) + c
y = 5*(x+1)
```


Popularity By Average Sentiment
Dish Average Sentiment
0
1
2
3
4
5
6
the indian sampler
south india
indian cuisine
ice cream
rogan josh
chick peas
dipping sauce
rice pudding
india pale
fried rice
gulab jamun
the naan was
flat bread
basmati rice
tomato soup
tikka masala
hot sauce
chicken tikka
chicken tikka masala
coconut chicken
tandoori chicken
mother india
tomato sauce
Dish Name

Task 4 : Discussion



	rank	A_dish_count	B_restaurant_count	C_avg_rating	D_avg_sentiment
0	1	tikka masala	chicken tikka	dipping sauce	the indian sampler
1	2	chicken tikka	tikka masala	ice cream	south india
2	3	chicken tikka masala	chicken tikka masala	the indian sampler	indian cuisine
3	4	tandoori chicken	indian cuisine	rogan josh	ice cream
4	5	indian cuisine	tandoori chicken	rice pudding	rogan josh
5	6	basmati rice	the naan was	flat bread	chick peas
6	7	south india	basmati rice	south india	dipping sauce
7	8	the naan was	gulab jamun	chick peas	rice pudding
8	9	rice pudding	rice pudding	indian cuisine	india gate
9	10	gulab jamun	south india	coconut chicken	fried rice
10	11	ice cream	ice cream	basmati rice	gulab jamun
11	12	rogan josh	rogan josh	gulab jamun	the naan was
12	13	dipping sauce	dipping sauce	chicken tikka	flat bread
13	14	hot sauce	hot sauce	tikka masala	basmati rice
14	15	tomato soup	chick peas	chicken tikka masala	tomato soup
15	16	fried rice	tomato sauce	tandoori chicken	tikka masala
16	17	india gate	fried rice	fried rice	hot sauce
17	18	chick peas	flat bread	mother india	chicken tikka
18	19	flat bread	tomato soup	tomato soup	chicken tikka masala
19	20	mother india	coconut chicken	the naan was	coconut chicken
20	21	tomato sauce	india gate	tomato sauce	tandoori chicken
21	22	coconut chicken	mother india	hot sauce	mother india
22	23	the indian sampler	the indian sampler	india gate	tomato sauce

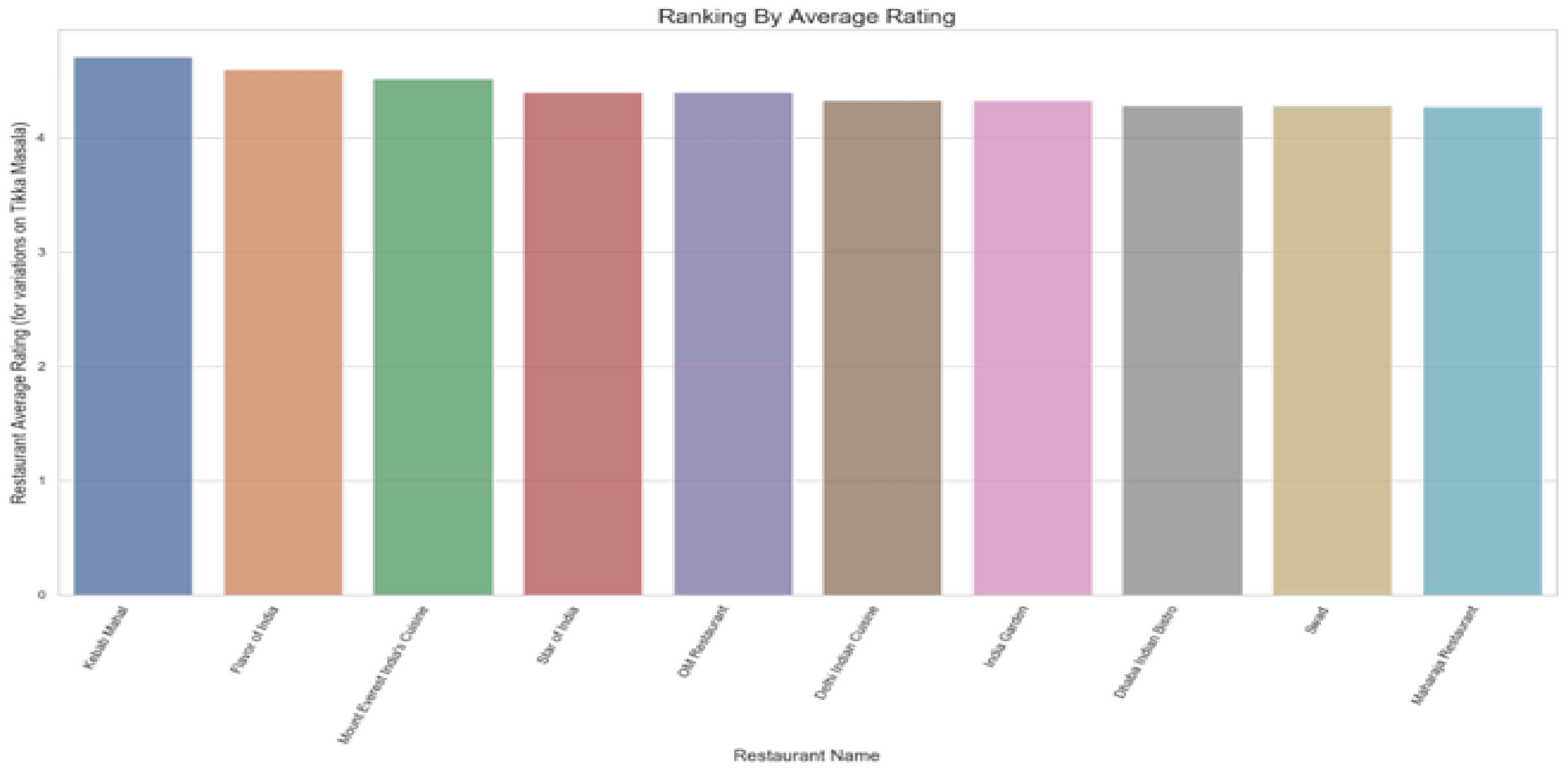
As shown in the comparison table, the count-based ranking system—whether applied at the rating or restaurant level—yields very similar results, which is expected. While the overall count decreases when considering only restaurants instead of mentions, the relative order of t he counts remains unchanged. However, the results diverge when considering average rating and average sentiment, as these metrics involve ranki ng different aspects.

Average rating focuses on the mean star rating given by users for a dish, while average sentiment seeks to identify dishes with the highest positive polarity scores. Ultimately, the concept of ranking popularity depends on how "popularity" is defined—whether it refers to higher ratings or more frequent mentions. An alternative approach could involve developing a custom algorithm that combines the four metrics explored. If I were to choose a method, I would likely prefer counting the total number of restaurants that mention a particular dish in their reviews, as I associate popularity with frequency (e.g., number of Instagram posts). In the context of Indian cuisine, dishes like chicken tikka masala, tandoori chicken, and naan come to mind as popular choices.

Task 5: Restaurant Ranking

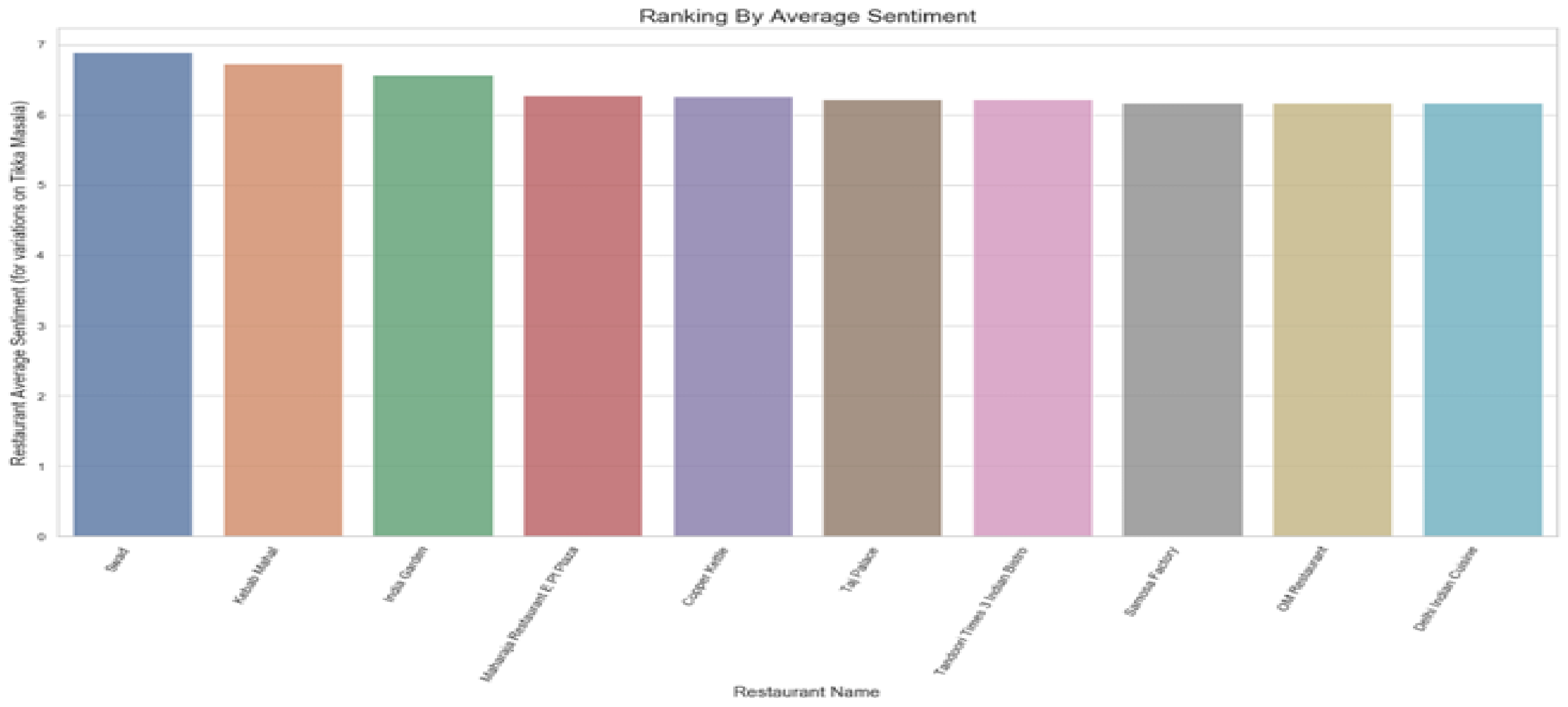
Task 5.1: Recommendation By Average Rating

A straightforward and easy-to-implement approach involves collecting all reviews mentioning a specific dish and calculating the average rating for each restaurant based on those reviews. The restaurant with the highest average rating for reviews containing the dish would be ranked at the top. In this case, we calculate the average ratings for our selected dishes by dividing the total ratings by the number of reviews that mention at least one of our chosen dishes (essentially all variations of tikka masala). To avoid division by zero, a small value is added to both the numerator and denominator when computing the average. After determining the average rating, the restaurants are sorted in descending order by their average rating, with review count used as a tiebreaker if necessary. Restaurants with fewer than five reviews are excluded, as a restaurant with a perfect rating from just one review may not be as reliable as one with a slightly lower rating but more reviews. Finally, the top 10 restaurants are selected for recommendation, as this is a common number of results returned by Yelp and other recommendation systems. As in Task 4, we use Seaborn's barplot for visualization.



Task 5.2: Popularity By Average Sentiment

In this approach, the same TextBlob package and ranking methodology from Task 4.4 are used to capture the average sentiment, but at the restaurant level. The scaled sentiment for each review mentioning tikka masala is calculated for each restaurant, and the sum of the scaled sentiments is divided by the total number of reviews for that restaurant. Reviews with a 3-star rating are excluded, and, similar to Task 5.1, only the top 10 restaurants are considered after filtering out those with fewer than five reviews.



	rank	A_avg_rating	B_avg_sentiment
0	1	Kebab Mahal	Swad
1	2	Flavor of India	Kebab Mahal
2	3	Mount Everest India's Cuisine	India Garden
3	4	Star of India	Maharaja Restaurant E Pt Plaza
4	5	OM Restaurant	Copper Kettle
5	6	Delhi Indian Cuisine	Taj Palace
6	7	India Garden	Tandoori Times 3 Indian Bistro
7	8	Dhaba Indian Bistro	Samosa Factory
8	9	Swad	OM Restaurant
9	10	Maharaja Restaurant	Delhi Indian Cuisine

Restaurant Ranking Discussion

As observed, there is some overlap in the top 10 recommended restaurants from both ranking methodologies (e.g., Swad and Kebab Mahal), but overall, the lists differ. Both ranking methods yield meaningful results, but the choice of which to use in an application may depend on the preferences of the end user. For example, an average rating ranking is simple, reliable, and works well for most users. Personally, when using Yelp, I tend to combine rating and price ($$) to decide where to dine, which could be another factor to explore. On the other hand, sentiment scores and polarity provide a useful alternative, as they capture a user's strong feelings about a particular restaurant and dish, something that the star rating system may not fully reflect. Therefore, more adventurous users might prefer this sentiment-based ranking system.

Task 6: Hygiene prediction

1. Overview

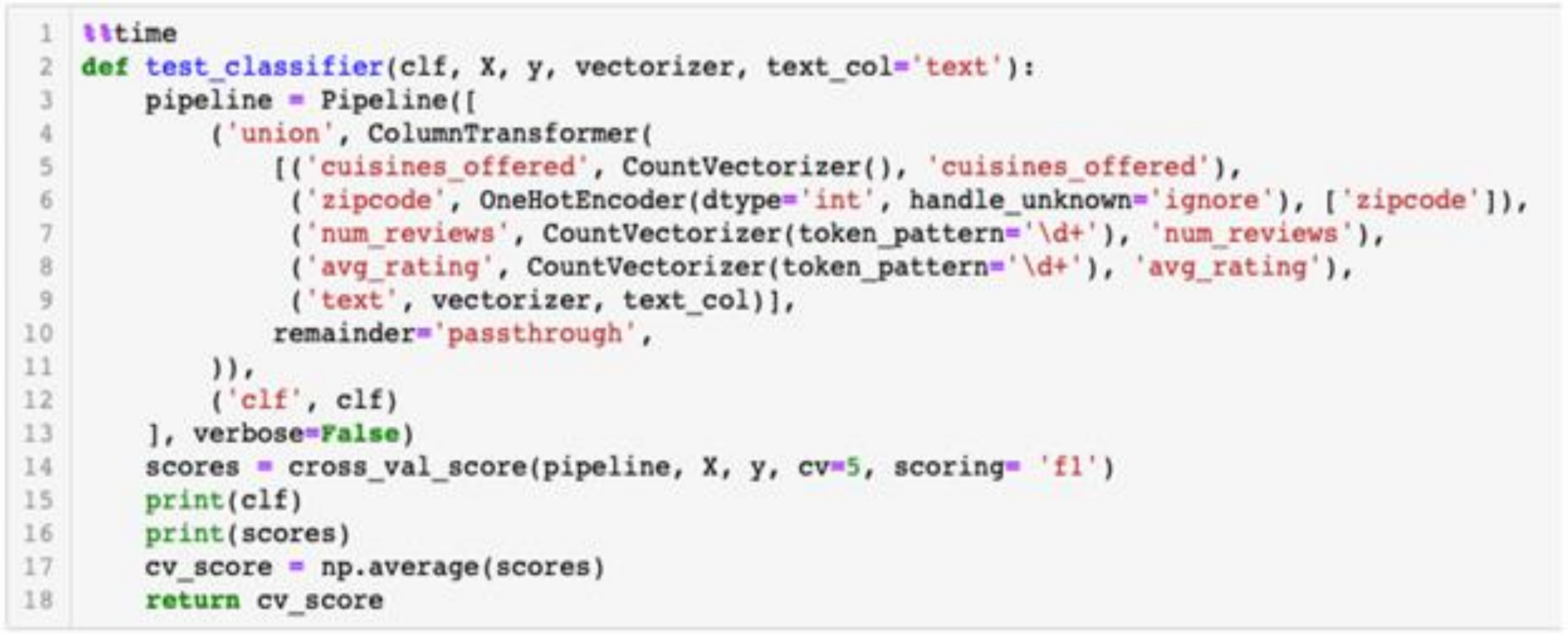
The aim of this task is to assist customers in choosing restaurants based on their hygiene standards, specifically whether they passed a hygiene inspection. The dataset includes concatenated text reviews of restaurants along with additional features such as cuisines offered, zip code, number of reviews, and average rating. It also contains labels indicating whether each restaurant has passed the latest public health inspection. As outlined in the problem statement, the first 546 entries represent labeled training data, while the remaining 12,753 entries are unlabeled and require label predictions.

2. Description and Comparison of Methods

2.1 Data Preprocessing

For data preprocessing, I applied many of the steps I had learned from Task 1 and used Gensim's preprocessing function with most of the default filters. This function converts text to lowercase, removes undesirable characters (e.g., tags, punctuation, and extra whitespaces), eliminates stop words, and stems the text to return a list of tokens. Additionally, I utilized NLTK's WordNet Lemmatizer to transform the tokenized words into their base forms. Overall, I believe these preprocessing steps effectively "normalize" the text, focusing on meaningful and significant words for input into language models and classifiers.

2.2 Text Representation/ Feature Engineering/Modeling



```
%%time
def test_classifier(clf, X, y, vectorizer, text_col='text'):
    pipeline = Pipeline([
        ('union', ColumnTransformer(
            [('cuisines_offered', CountVectorizer(), 'cuisines_offered'),
             ('zipcode', OneHotEncoder(dtype='int', handle_unknown='ignore'), ['zipcode']),
             ('num_reviews', CountVectorizer(token_pattern='\d+'), 'num_reviews'),
             ('avg_rating', CountVectorizer(token_pattern='\d+'), 'avg_rating'),
             ('text', vectorizer, text_col)],
            remainder='passthrough',
        )),
        ('clf', clf)
    ], verbose=False)
    scores = cross_val_score(pipeline, X, y, cv=5, scoring= 'f1')
    print(clf)
    print(scores)
    cv_score = np.average(scores)
    return cv_score
```

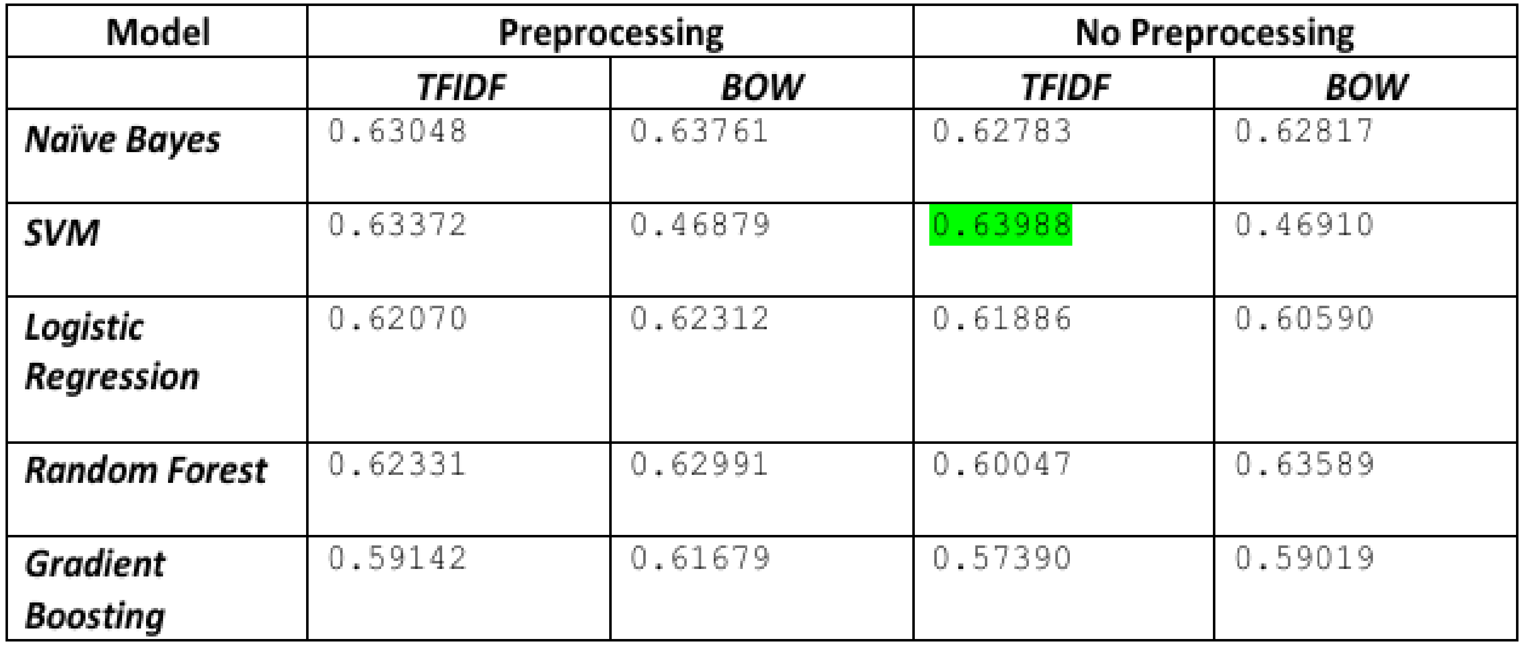
My first priority was to design a process that made it easy to test different text representations and machine learning models for the classification task. Fortunately, sklearn pipelines helped streamline the workflow by enabling efficient feature transformation and model evaluation. The ColumnTransformer was particularly helpful as it allowed me to specify various feature columns of different types and apply tailored transformations to each. For instance, I used OneHot Encoding for the 30 zip codes in the dataset and CountVectorizer() to represent different cuisine types as features.

The text column, being a critical feature, was primarily represented using a Bag of Words approach with CountVectorizer() and TfidfVectorizer(). In general, TfidfVectorizer outperformed CountVectorizer since it not only counted word frequencies but also adjusted word importance using inverse document frequency. While I briefly experimented with word2vec embeddings, they did not significantly improve performance.

The sklearn pipeline proved invaluable, as it enabled the seamless combination of column transformations with the addition of a classifier as a secondary step. For comparison, I implemented several models in sklearn, including Naïve Bayes, Support Vector Machine (SVM), Random Forest, and Gradient Boosting. Although selecting the right model is crucial—SVM and Naïve Bayes are particularly popular for text classification with limited training data—I believe the most critical aspect of this task was feature engineering, especially for text representation.

2.3 Methods Comparison

For most problems, I would usually just use train_test_split in order to split the training data into roughly 80% for training and 20% for validation. However, because we know that because our dataset is imbalanced, it is better to have a cross-validation strategy. Thus, for evaluation purposes, I used sklearn’s cross_val_score function in order to perform cross-validation of model performance across 5 distinct folds. I believe that this is a better approach because the models may perform very differently on different data splits. In the next slide, a table showing the results.



Model	Preprocessing		No Preprocessing	
	TFIDF	*BOW*	*TFIDF*	*BOW*
Naïve Bayes	0.63048	0.63761	0.62783	0.62817
SVM	0.63372	0.46879	0.63988	0.46910
Logistic Regression	0.62070	0.62312	0.61886	0.60590
Random Forest	0.62331	0.62991	0.60047	0.63589
Gradient Boosting	0.59142	0.61679	0.57390	0.59019

Due to the differences in data distribution between the training dataset and the hidden leaderboard dataset, it was challenging to accurately compare the performance of my model attempts. For example, some models that performed well locally did not necessarily achieve a higher F1 score on the leaderboard. This suggests that tuning needs to be specific to the given dataset. According to the No Free Lunch Theorem, every model relies on specific assumptions about the data, and certain models will perform better than others in particular scenarios.

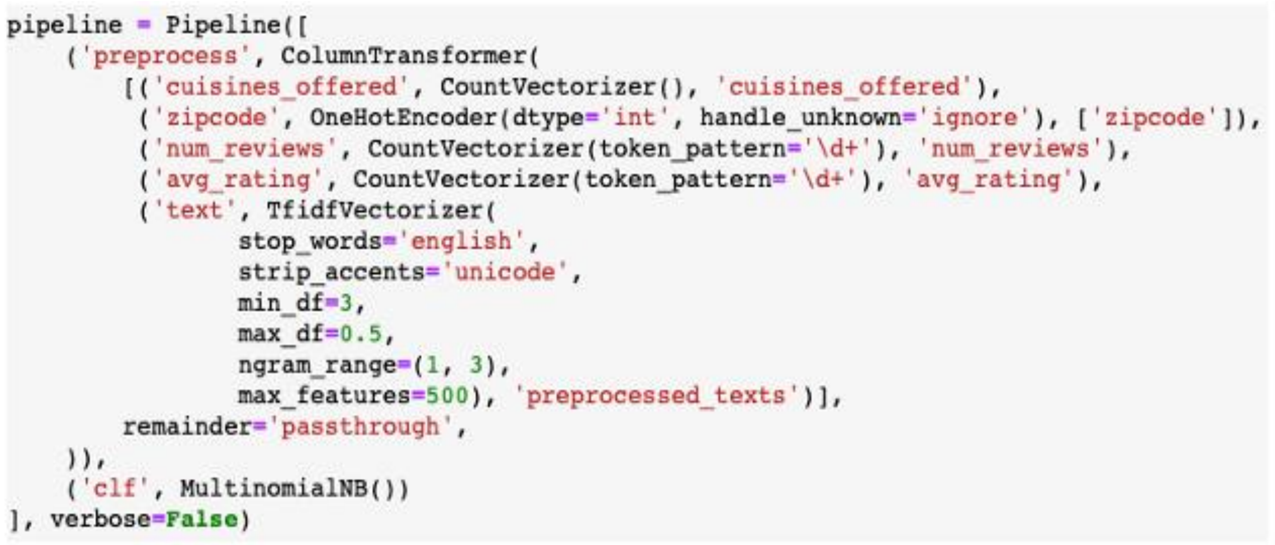
Overall, SVM and Naïve Bayes generally performed better on the leaderboard, likely because they have a strong track record in text classification and are less prone to overfitting compared to other models.

2.4 Ensembling

I attempted to create an ensemble of the various models I experimented with using a package called mlens. The approach involved using multiple models (e.g., Random Forest, Logistic Regression, Naïve Bayes, etc.) as base learners and passing their outputs to a second-layer meta-learner, which blends their predictions by learning from the base models' class predictions. In theory, this method should work well because the diversity of uncorrelated base learners can help the meta-learner address the individual weaknesses of each model. As shown in the table, many of the cross-validated scores appeared to be largely uncorrelated.

With more time, I would have liked to further explore ensembling techniques, particularly by fine-tuning the final meta layer. At the time of this report, I only used a simple Logistic Regression classifier as the meta-learner, which did not lead to improved leaderboard performance.

My highest-performing model on the leaderboard achieved a score of 0.7201, securing 24th place at the time. However, when evaluated locally, it achieved an average F1 score of only 0.63043 across five folds.



```
pipeline = Pipeline([
    ('preprocess', ColumnTransformer(
        [('cuisines_offered', CountVectorizer(), 'cuisines_offered'),
         ('zipcode', OneHotEncoder(dtype='int', handle_unknown='ignore'), ['zipcode']),
         ('num_reviews', CountVectorizer(token_pattern='\d+'), 'num_reviews'),
         ('avg_rating', CountVectorizer(token_pattern='\d+'), 'avg_rating'),
         ('text', TfidfVectorizer(
                stop_words='english',
                strip_accents='unicode',
                min_df=3,
                max_df=0.5,
                ngram_range=(1, 3),
                max_features=500), 'preprocessed_texts')],
        remainder='passthrough',
    )),
    ('clf', MultinomialNB())
], verbose=False)
```

4. Future Work

An attempt was made to train a Word2Vec model using Gensim's Word2Vec interface to generate word embeddings. The resulting feature vectors (with a chosen size of 300) were then passed to an XGBoost classifier, yielding a local score of 0.62015. However, this did not provide a significant improvement over the TF-IDF feature representation, and further exploration was limited due to time constraints. Given more time, additional word embeddings, such as pretrained GloVe or FastText embeddings, could be tested to evaluate their effectiveness in representing text. Exploring deep learning model architectures, including a simple feed-forward neural network or LSTM trained on these embeddings, would also be an interesting avenue for improvement.

Beyond model experimentation, there is a strong interest in developing a more robust pipeline for text classification experiments. Integrating word embeddings into an existing scikit-learn pipeline and column transformer proved to be non-trivial. A deeper exploration of incorporating a deep learning model at the end of the pipeline, instead of relying on standard scikit-learn classifiers, would be beneficial.

Additionally, optimizing hyperparameters through GridSearchCV could improve model performance by identifying the best parameters for the chosen classifiers. Addressing class imbalance through techniques such as undersampling or adjusting class weights may also enhance results. These improvements could potentially lead to better leaderboard performance.

References
• https://scikit learn.org/stable/auto_examples/compose/plot_column_transformer.html#sphx-glr auto-examples-compose-plot-column-transformer-py – example for column transformer
• https://radimrehurek.com/gensim/parsing/preprocessing.html - preprocessing library used • https://www.kaggle.com/baghern/a-deep-dive-into-sklearn-pipelines - great resource on pipelining
• https://towardsdatascience.com/nlp-performance-of-different-word-embeddings-on text-classification-de648c6262b -
• https://mlens.readthedocs.io/en/0.1.x/getting_started/ - reference for introductory ensembling
• https://www.kdnuggets.com/2019/09/no-free-lunch-data-science.html - No Free Lunch Theorem