Final Project Report

Item-based Collaborative Recommendation Using Embeddings for Establishments from the Yelp Dataset

Mai Ngo and Jenish Dobariya

CSC 575 Intelligent Information Retrieval - SEC 801

Dr. Noriko Tomuro

March 20, 2024

Table of Contents

Task 1 - Business Recommendation by Table Lookup

We did Task 1 independently with the objective is to examine the data, especially data types and gain a good intuition on how to approach the project. Overall, we noticed three columns ‘attributes’, ‘city’ and ‘categories’ are dominant in determining recommendations. Then tried different approaches to correctly extract values from these columns just based on manual user query. ‘city’ was easy, however, ‘attributes’ and ‘categories’ needs some further text pre-processing on the data side.

Ultimately, we decided to transform the text in ‘categories’ column to individual words. For example, **‘Shipping Centers’ to ‘ShippingCenters’** | **‘Coffee & Tea’ to ‘Coffee&Tea’**. At this point we believed this format is appropriate to retrieve the right categories based on user queries. However, later on in Task 3 we noticed that this was not a good approach which will be discussed later.

Functions getRec and geoRec

This was an extra step we took to fully understand the Businesses dataset, the function is pretty long but by successfully coding it, we definitely gained a good knowledge in which establishment should be recommended logically. Also easier for us to approach Task 3 form both user and coder perspectives.

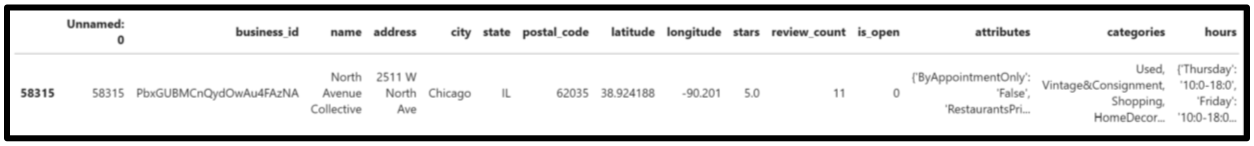
The function’s goal is to get recommendations from users queries, basically mimicking the manual cross-tabulation. We want to make sure it is reusable with all queries, so lots of text processing on the query’s side. The accepting query’s template is **‘Best Categories1 Categories2 in City that have/has/are Attribute’** given **‘Best’** and **Categories1** are optional.

We also decided that **‘Best’** pertains to establishment with star ratings of at least 4.0, **‘queryRes = queryRes[queryRes['stars'] > 4.0]’**. The establishment’s category like ‘Restaurant’ will be searched on the ‘categories’ column, with string contains ‘Restaurants’, case sensitive. Overall, we satisfied with Task 1 output, able to generate geospatial map with all recommendations using library ‘folium’, as well ‘geopy’ for city’s name as center of the map. Additionally, function **geoRec(query, numRec)** calls function **getRec** to get final output.

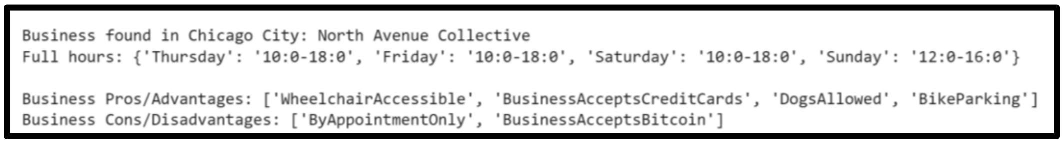
<u>**Note:**</u> Geocoding service Nominatim is very sensitive to connection, it will give error right away if connection to the server cannot be established. The code should work, if not need to run the cell couple times.

Follow-up Query

We started out tested with the example query **‘Business hours for "DeSandro on Main" in Philadelphia for Friday’**. Then our own query search is **‘Business hours and Pros/Cons for "North Avenue Collective" in Chicago’**. We chose this just because of 'Chicago', as we want to look up businesses in the city and found out there is only one Chicago business in this dataset. Pros and cons are decided based on the **‘attributes’** column. Below is the output:



	Unnamed: 0	business_id	name	address	city	state	postal_code	latitude	longitude	stars	review_count	is_open	attributes	categories	hours
58315	58315	PbxGUBMCnQydOwAu4FAzNA	North Avenue Collective	2511 W North Ave	Chicago	IL	62035	38.924188	-90.201	5.0	11	0	{'ByAppointmentOnly': 'False', 'RestaurantsPri...	Used, Vintage&Consignment, Shopping, HomeDecor...	{'Thursday': '10:0-18:0', 'Friday': '10:0-18:0...

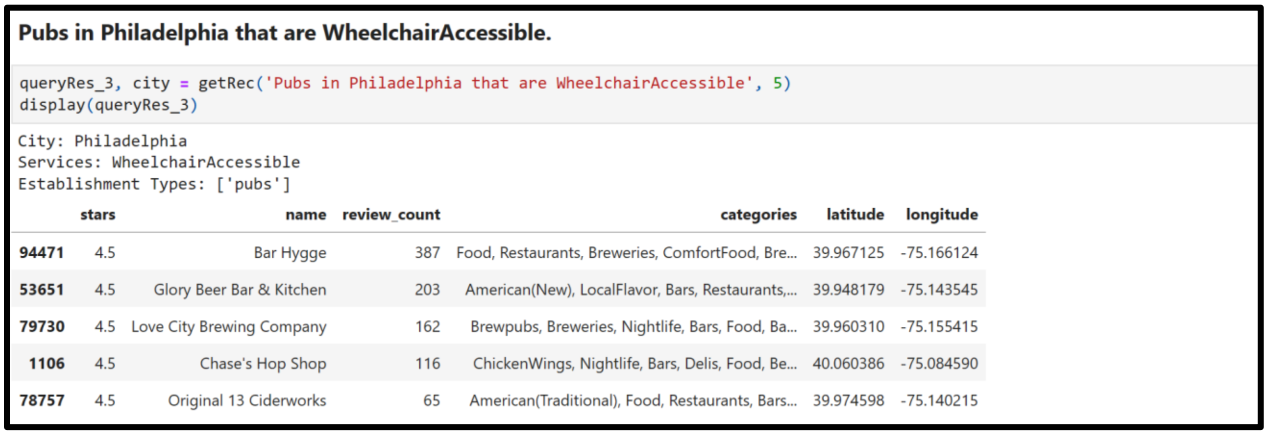


```
Business found in Chicago City: North Avenue Collective
Full hours: {'Thursday': '10:0-18:0', 'Friday': '10:0-18:0', 'Saturday': '10:0-18:0', 'Sunday': '12:0-16:0'}

Business Pros/Advantages: ['WheelchairAccessible', 'BusinessAcceptsCreditCards', 'DogsAllowed', 'BikeParking']
Business Cons/Disadvantages: ['ByAppointmentOnly', 'BusinessAcceptsBitcoin']
```

Output Accuracies

First, we need to make sure our getRec function runs correctly. So, we cross-checked using sample queries, outputs from both manual and function approaches are the same.

<u>*Function output:*</u>



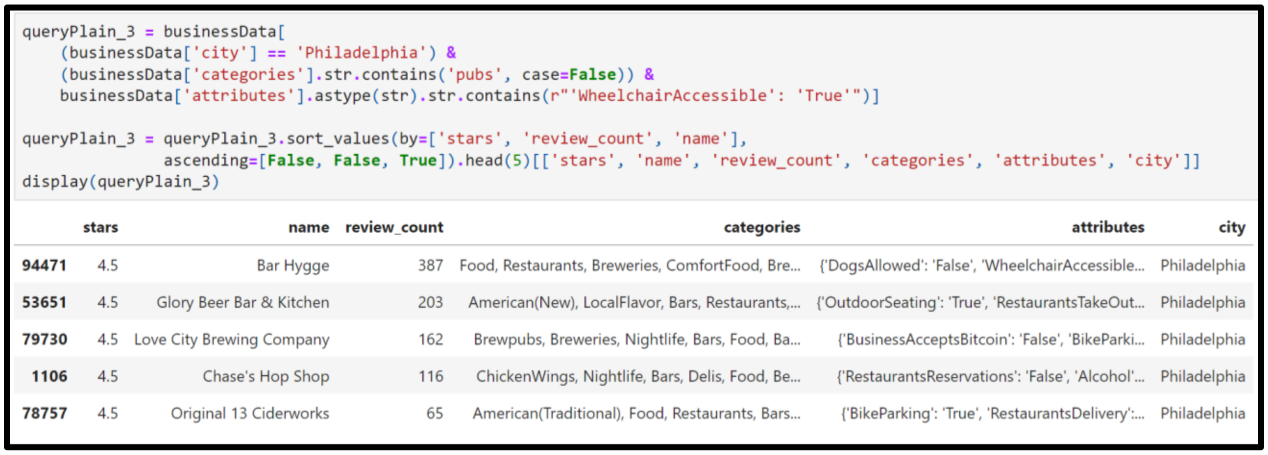
Pubs in Philadelphia that are WheelchairAccessible.

```
queryRes_3, city = getRec('Pubs in Philadelphia that are WheelchairAccessible', 5)
display(queryRes_3)
```

```
City: Philadelphia
Services: WheelchairAccessible
Establishment Types: ['pubs']
```

	stars	name	review_count	categories	latitude	longitude
94471	4.5	Bar Hygge	387	Food, Restaurants, Breweries, ComfortFood, Bre...	39.967125	-75.166124
53651	4.5	Glory Beer Bar & Kitchen	203	American(New), LocalFlavor, Bars, Restaurants,...	39.948179	-75.143545
79730	4.5	Love City Brewing Company	162	Brewpubs, Breweries, Nightlife, Bars, Food, Ba...	39.960310	-75.155415
1106	4.5	Chase's Hop Shop	116	ChickenWings, Nightlife, Bars, Delis, Food, Be...	40.060386	-75.084590
78757	4.5	Original 13 Ciderworks	65	American(Traditional), Food, Restaurants, Bars...	39.974598	-75.140215

Manual query output:



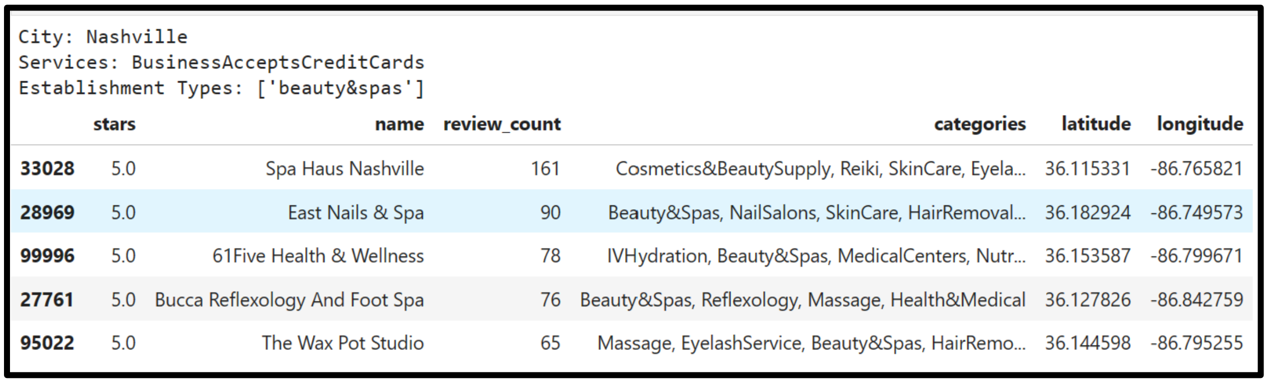
```
queryPlain_3 = businessData[
    (businessData['city'] == 'Philadelphia') &
    (businessData['categories'].str.contains('pubs', case=False)) &
    businessData['attributes'].astype(str).str.contains(r"'WheelchairAccessible': 'True'")]

queryPlain_3 = queryPlain_3.sort_values(by=['stars', 'review_count', 'name'],
               ascending=[False, False, True]).head(5)[['stars', 'name', 'review_count', 'categories', 'attributes', 'city']]
display(queryPlain_3)
```

	stars	name	review_count	categories	attributes	city
94471	4.5	Bar Hygge	387	Food, Restaurants, Breweries, ComfortFood, Bre...	{'DogsAllowed': 'False', 'WheelchairAccessible...	Philadelphia
53651	4.5	Glory Beer Bar & Kitchen	203	American(New), LocalFlavor, Bars, Restaurants,...	{'OutdoorSeating': 'True', 'RestaurantsTakeOut...	Philadelphia
79730	4.5	Love City Brewing Company	162	Brewpubs, Breweries, Nightlife, Bars, Food, Ba...	{'BusinessAcceptsBitcoin': 'False', 'BikeParki...	Philadelphia
1106	4.5	Chase's Hop Shop	116	ChickenWings, Nightlife, Bars, Delis, Food, Be...	{'RestaurantsReservations': 'False', 'Alcohol'...	Philadelphia
78757	4.5	Original 13 Ciderworks	65	American(Traditional), Food, Restaurants, Bars...	{'BikeParking': 'True', 'RestaurantsDelivery':...	Philadelphia

Second, 5 queries that we came up with shown below. Output are recommendations list along with geo map using the city's center as benchmark:

1. Health&Medical in Reno that are AcceptsInsurance
2. Best Beauty&Spas in Nashville that are BusinessAcceptsCreditCards
3. FastFood Restaurants in Ashland City
4. Best FastFood Restaurants in Ashland City
5. Italian Restaurants in Affton that do Caters

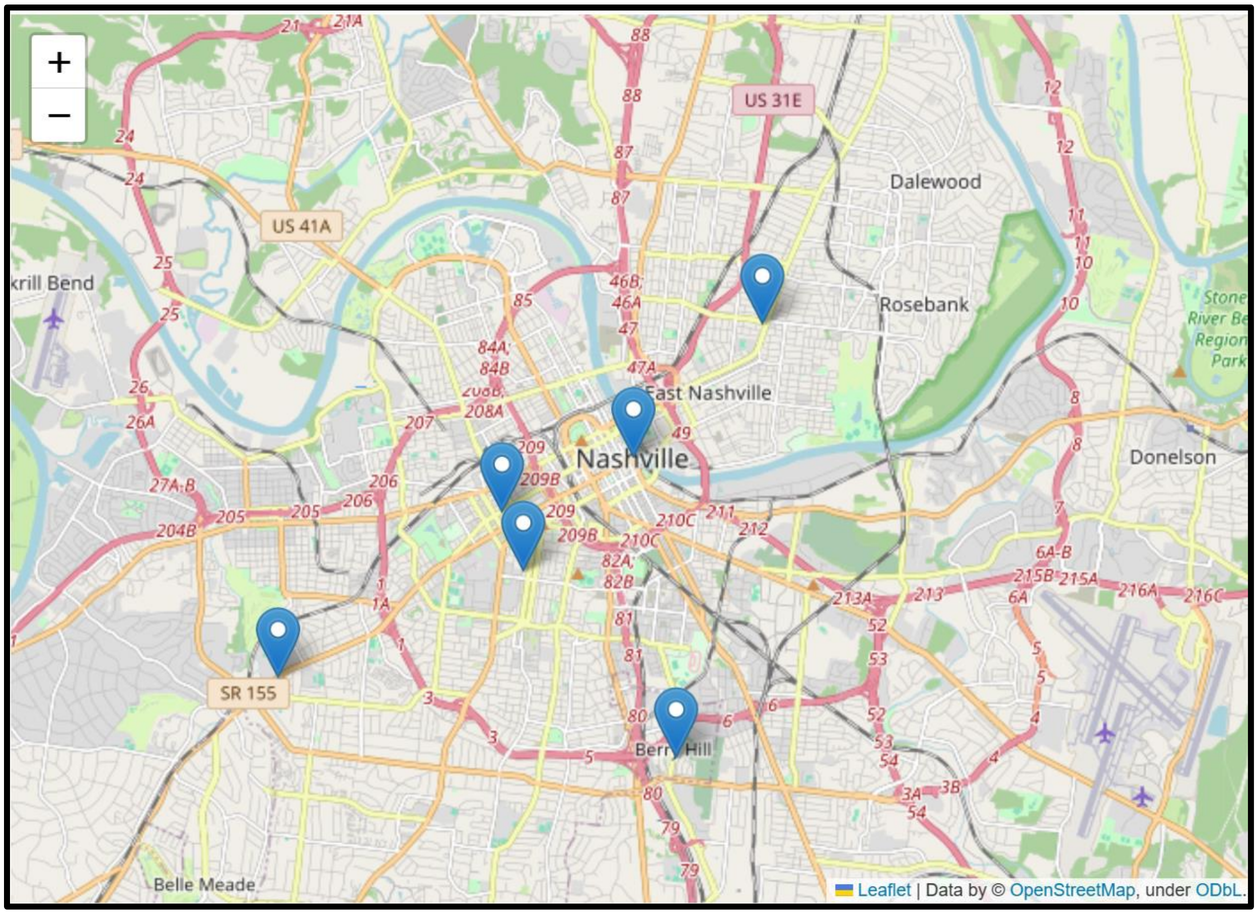
Output from query **'Best Beauty&Spas in Nashville that are BusinessAcceptsCreditCards'**:



```
City: Nashville
Services: BusinessAcceptsCreditCards
Establishment Types: ['beauty&spas']
```

	stars	name	review_count	categories	latitude	longitude
33028	5.0	Spa Haus Nashville	161	Cosmetics&BeautySupply, Reiki, SkinCare, Eyela...	36.115331	-86.765821
28969	5.0	East Nails & Spa	90	Beauty&Spas, NailSalons, SkinCare, HairRemoval...	36.182924	-86.749573
99996	5.0	61Five Health & Wellness	78	IVHydration, Beauty&Spas, MedicalCenters, Nutr...	36.153587	-86.799671
27761	5.0	Bucca Reflexology And Foot Spa	76	Beauty&Spas, Reflexology, Massage, Health&Medical	36.127826	-86.842759
95022	5.0	The Wax Pot Studio	65	Massage, EyelashService, Beauty&Spas, HairRemo...	36.144598	-86.795255

Geo map:


Dalewood
Rosebank
East Nashville
Nashville
Donelson
Berry Hill
Belle Meade
US 41A
US 31E
SR 155
Leaflet | Data by © OpenStreetMap, under ODbL.

Task 2 - Recommender System using Embeddings

Step 1: Create Embeddings

Following a thorough cleansing process facilitated by spaCy, a widely respected library renowned for its efficacy in real-world applications, our data underwent meticulous preparation for embedding computation. At the heart of this endeavor lies the **`process_txt`** function, an indispensable component of our data preprocessing pipeline, meticulously engineered to filter textual data with precision. By implementing various conditions, this function adeptly removes stop words, eliminates excessively lengthy or truncated words, and validates the alphanumeric nature of tokens, ultimately yielding a refined list of tokens impeccably groomed and optimized for embedding purposes.

Upon procuring the sanitized text tokens, as they were assimilated into a fresh column mapped to our training dataset. Next, we segmented of our training data into two distinct data frames, grouped by user ID and business ID, respectively, served to tailor embeddings specific to individual users and businesses, ensuring a granular understanding of their unique characteristics.

We harnessed the **`load_glove`** function, meticulously configured to load embeddings of precisely 50 dimensions, calibrated to match the requirements of our project. This methodical approach ensures that our embeddings accurately capture the nuanced intricacies inherent in our dataset.

Moreover, we engineered a **`compute_mean_embedding`** function, designed to process a list of tokens and retrieve corresponding embeddings from the GloVe dictionary. Through this process, the function calculates the mean embedding. This iterative application extends to both the 6 user ID and business ID data frames, ensuring comprehensive embedding coverage across our dataset.

Step 2: Recommendations (Rating Predictions)

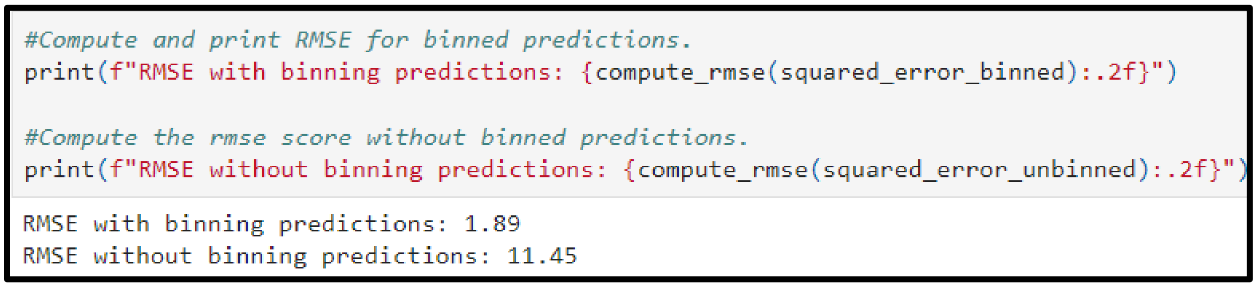
We pivot our focus towards developing functions to compute mean squared error and root mean squared error. In executing this task, we traverse the testing dataset, meticulously validating both user ID and business ID within our user and business embedding tables, respectively. Should either identifier be absent, the computation is bypassed for that particular data row, ensuring integrity and accuracy in subsequent analyses.

For eligible data rows where both user ID and business ID are found within the corresponding embedding tables, embeddings are retrieved and employed to calculate the dot product. This pivotal computation serves as the cornerstone for deriving mean square error and root mean squared error scores, utilizing the ground truth ratings provided in the testing data frame.

Upon completing the computation of dot products for both the testing and training datasets, we proceeded with the task of predicting ratings for individual businesses. However, a notable discrepancy emerged between the dot product values and the actual rating values, resulting in a notable elevation in the root mean squared error (RMSE) score. To address this discrepancy and enhance the accuracy of our rating predictions derived from embeddings, we employed a binning strategy.

By discretizing the dot product values into distinct intervals, carefully aligned with the range of star ratings, we aimed to ensure a more accurate correspondence between computed dot product values and the expected rating ranges. This strategic adjustment allowed for a more nuanced interpretation of the embedding-based predictions, mitigating the observed disparity and improving the overall fidelity of our rating estimation framework.

Following the implementation of binning, we observed a tangible improvement in predictive performance, as evidenced by a substantial reduction in the RMSE to a commendable 1.89. This outcome underscores the effectiveness of our intervention, demonstrating the importance of strategic adjustments in refining predictive modeling methodologies and optimizing analytical outcomes.



```
#Compute and print RMSE for binned predictions.
print(f"RMSE with binning predictions: {compute_rmse(squared_error_binned):.2f}")

#Compute the rmse score without binned predictions.
print(f"RMSE without binning predictions: {compute_rmse(squared_error_unbinned):.2f}")
```

```
RMSE with binning predictions: 1.89
RMSE without binning predictions: 11.45
```

Task 3 - Item-based Collaborative Recommendation using Embeddings

Task 3 is a combination of Task 1 and 2. At first, we thought we can just reuse the code from Task 1, however, after some trials and errors, we decided to reformat the 'categories' column more efficient so that values can be retrieved optimally. Function **'splitCategories'** will convert cell values in 'categories' column into a list of strings. For example:

'Doctors, Traditional Chinese Medicine, Naturopathic/Holistic, Acupuncture, Health & Medical, Nutritionists' will be converted to **[Doctors, Traditional, Chinese, Medicine, Naturopathic, Holistic, Acupuncture, Health, Medical, Nutritionists]**

We also tackle special cases like **'Coffee & Tea'** to individual strings **'Coffee'** and **'Tea'.** This makes much more sense to find 'matching' businesses for recommendation, this approach's usefulness will be discussed further later.

Discussing further our setup for this recommendation system. Users will enter a query that contains an establishment's name (optionally) with a city name (mandatory) that they want to generate recommendations on. With this, if the user not entering establishment's name as a benchmark, we will use business categories mentioned in the query to filter. If both business categories and establishment's name are entered, we will filter both. In order to create an efficient recommendation algorithm, we create two functions **'filteredData'** and **'computeRecommendation'**.

Function filteredData

This function's objective is to filter **'businesses'** data that only pertains to the requirements in user's query. Noticeably, **'Best'** logic is not accounted here per project guidance. Overall, the function is pretty long but we think it is efficient. Start with applying tokenization on the query to extract necessary information. Here are basic logic:

1. Extract **city name** using spaCy, location strings are detected with 'GPE' label tag. We also leaves room if the user enters multiple cities, which strings are appended to 'cityList'.

2. Extract **establishment's name**, note business name will need to be quoted in quotation marks. Regular expression is used to extract business name stored as 'placeName'. Then extract categories list of the establishment name stored as **'categories'**
3. Extract **business categories**, this is just tokens list from user query stored as 'tokenList'.

<u>The code will filter data based on two scenarios:</u>

A. User enters **city name, establishment's name,** and **business categories.**

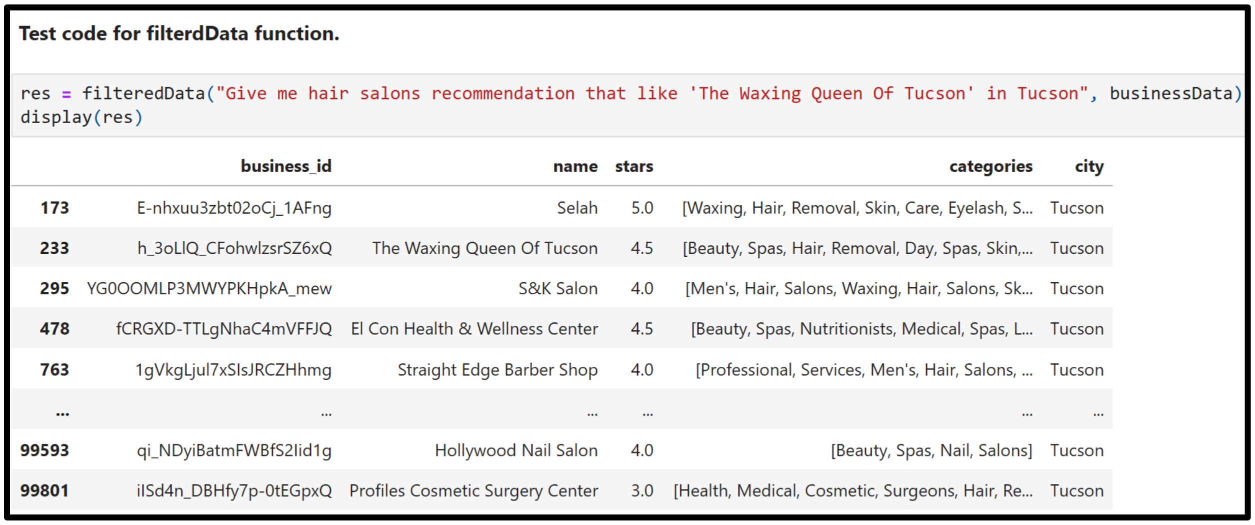
1) Given user did put a benchmark establishment, filtered businesses that must match at least 3 categories of benchmarking establishment. This to ensure the filtered business has same characteristics as benchmarking establishment.
2) Extra filter using categories they mentioned in the query 'tokenList'.
3) Filter only businesses in the city user requests.

B. User enters **city name and business categories.**

Same process but skipping Step 1.

<u>Note:</u> We leave room for further algorithm improvement. If user did not enter a city, output everything after filtering categories. Furthermore, by applying transformation on 'categories' column, implementation of our logic was very doable.

<u>Output Test from filteredData:</u>



Test code for filterdData function.

```
res = filteredData("Give me hair salons recommendation that like 'The Waxing Queen Of Tucson' in Tucson", businessData)
display(res)
```

	business_id	name	stars	categories	city
173	E-nhxuu3zbt02oCj_1AFng	Selah	5.0	[Waxing, Hair, Removal, Skin, Care, Eyelash, S...	Tucson
233	h_3oLlQ_CFohwlzsrSZ6xQ	The Waxing Queen Of Tucson	4.5	[Beauty, Spas, Hair, Removal, Day, Spas, Skin,...	Tucson
295	YG0OOMLP3MWYPKHpkA_mew	S&K Salon	4.0	[Men's, Hair, Salons, Waxing, Hair, Salons, Sk...	Tucson
478	fCRGXD-TTLgNhaC4mVFFJQ	El Con Health & Wellness Center	4.5	[Beauty, Spas, Nutritionists, Medical, Spas, L...	Tucson
763	1gVkgLjul7xSIsJRCZHhmg	Straight Edge Barber Shop	4.0	[Professional, Services, Men's, Hair, Salons, ...	Tucson
...	...	...	...	...	...
99593	qi_NDyiBatmFWBfS2lid1g	Hollywood Nail Salon	4.0	[Beauty, Spas, Nail, Salons]	Tucson
99801	ilSd4n_DBHfy7p-0tEGpxQ	Profiles Cosmetic Surgery Center	3.0	[Health, Medical, Cosmetic, Surgeons, Hair, Re...	Tucson

Function computeRecommendation

The function will call and use data output from the ‘filteredData’ function. Same logic, first we apply tokenization on the query using function **‘process_text’** from Task 2, this to ensure words are processed in the same manner and get the mean embedding of the query.

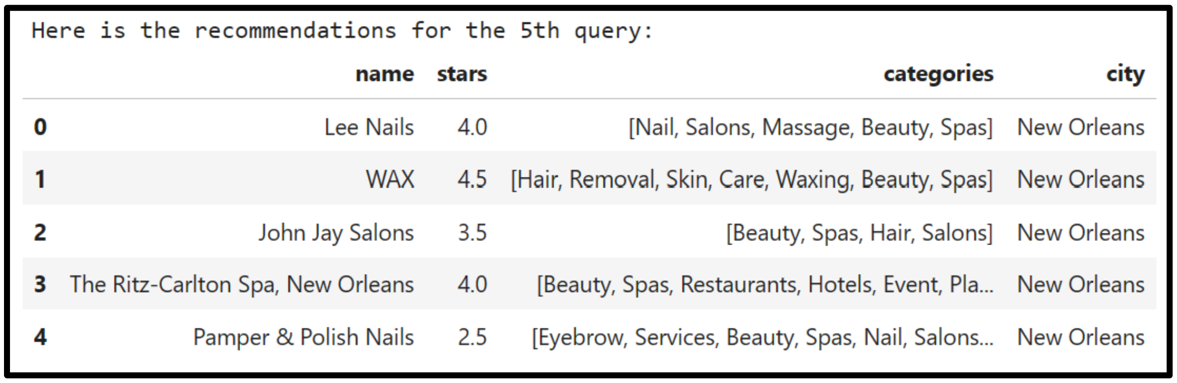
As for the data, we retrieve mean embeddings of the businesses existed in the filtered data, then calculate Cosine similarities with the query’s mean embeddings. Certainly, top recommendations are corresponding to high Cosine similarities. We also leave room that if there is no matching business between filtered data and business mean embedding data, the system would return as no recommendation within required city.

Output

Five queries that we came up with shown below. Output are recommendations list along with geo map using the city’s center as benchmark:

1. List sushi Japanese restaurants that are similar to 'Tuna Bar' in Santa Barbara
2. Give me IV Hydration spas recommendation in Nashville that similar to '61Five Health & Wellness'
3. Show me apartments complex like 'Ridge Carlton Apartments' in Indianapolis
4. Give me recommendation like 'Vintage Bar and Grill' in Tampa
5. Give me hair salons recommendation that like 'The Waxing Queen Of Tucson' in New Orleans

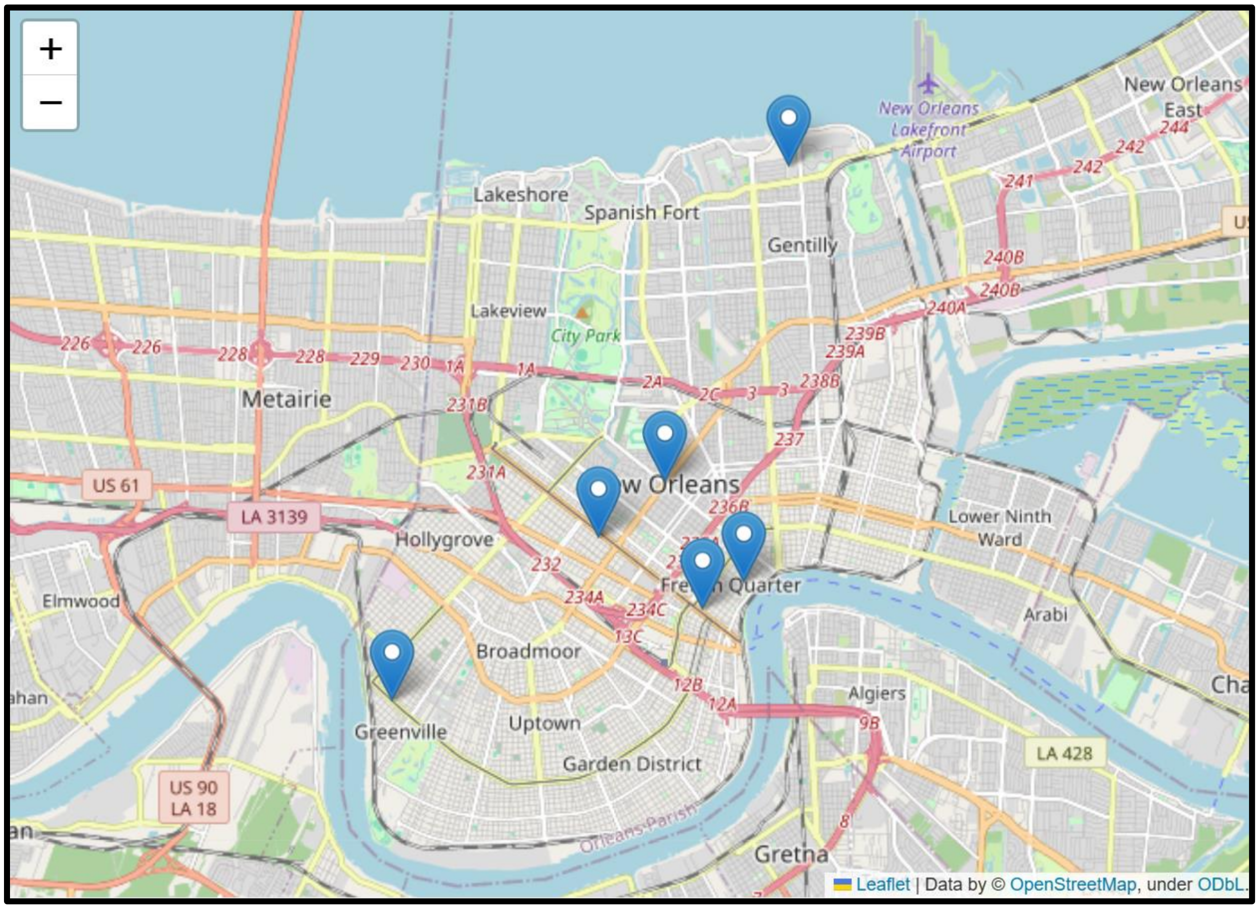
Output from query **Give me hair salons recommendation that like ‘The Waxing Queen Of Tucson' in New Orleans’**:



```
Here is the recommendations for the 5th query:
```

	name	stars	categories	city
0	Lee Nails	4.0	[Nail, Salons, Massage, Beauty, Spas]	New Orleans
1	WAX	4.5	[Hair, Removal, Skin, Care, Waxing, Beauty, Spas]	New Orleans
2	John Jay Salons	3.5	[Beauty, Spas, Hair, Salons]	New Orleans
3	The Ritz-Carlton Spa, New Orleans	4.0	[Beauty, Spas, Restaurants, Hotels, Event, Pla...	New Orleans
4	Pamper & Polish Nails	2.5	[Eyebrow, Services, Beauty, Spas, Nail, Salons...	New Orleans

Geo map:



Overall, we are very satisfied with Task 3 output, this is exactly what we wanted with the knowledge, trials and errors from Task 1 and 2. We want to emphasize that it took us many attempts and lots of time to code this system correctly, have to see it from both user and coder perspectives. Initially, we use 'city' as the benchmark, however, the algorithm was not giving output like what we conceptualized, so we re-grouped and decided to try using 'categories' and it worked.

Furthermore, successful integration between Task 1 and 2 was also challenging, not much on the coding side. We make sure thoroughly setup the algorithm on 'paper' first, ensuring it makes sense logically from user perspective, then start implementing code along the way. Eventually, some functions need light modification here and there to make sure joined code works, and we are very happy that our idea successfully implemented.

Reflection

Throughout this project, we really enjoy the conceptualization part. The coding was challenging, however, once we have the system setup correctly in our head, we just follow that structure slowly step-by-step.

As we have been working together before, it was not hard at all for us to do this project together. We worked on Task 1 and 2 independently, but make sure re-group occasionally to touch base on our logic as well asking for feedback from the other if our approach was right. To clarify, we did not just finish our task and let the other person carry the next step on their own. Every logic was discussed and explained thoroughly, we are very confident in each other's ability to code so that would never be the problem. Another thing that helped us is that our thinking and coding are somewhat same style. So, it does not take us long to explain something or understand the other person's code.

Furthermore, we also have the same thinking that we like to leave room for algorithm improvement. We like to mind map alternative approaches, implementation if it works so that later we can come back and develop further if we want to. The project guidance shows that Task 1 is pretty simple, heavy on Task 2, and merge on Task 3. However, to us, we carry each task as equally important, putting extra work on Task 1 and think different embedding approaches on Task 2, that is how we can get good understanding and source code to implement Task 3.

With respect to the course, have learned a lot throughout the assignment, and even more with this project. The final project allows us to have logical freedom which we both think it is the best way to enhance our programming skill and gain experience in creating a functional recommendation system. To sum up, we are very satisfied with this project. It was not what we initially visualized, even better. Due to time constraints, we have not tried to deploy it yet, but that would be a suggestion to further improve this project.

Member Contributions

- ✓ Task 1 – Mai Ngo
- ✓ Task 2 – Jenish Dobariya
- ✓ Task 3 – Mai Ngo and Jenish Dobariya