410 Project Progress Report

Team Name: DuoDuo

Project Topic: Restaurant Concierge

Team Members:

Rhea Chen	xinyuc11	xinyuc11@illinois.edu (captain)
Jingjing Yao	jyao27	jyao27@illinois.edu
Xian Chen	xianc2	xianc2@illinois.edu

Progress Made

Our project is about developing a tourist-oriented restaurant concierge service tailored for travelers with limited time in a city. The primary focus has been on designing a user-friendly command-line interface to streamline user interaction with our service.

By using the Yelp Open dataset, we successfully implemented the BM25 algorithm to rank restaurants, providing users with a curated list of top dining options based on their location. The main task of designing and working on the ranking algorithm has been nearly completed.

Simultaneously, the second task involving the construction of the Python application is also nearing completion, with a simple yet efficient command line interface that prompts users for location input and delivers the top 10 recommended restaurants. Our team is excited about the progress made and looks forward to finalizing testing, report writing, and presenting the accomplished results.

To enhance transparency and understanding about the progress in algorithm development and application construction, we provide some screenshots of our codebase and the code result at the end of the report for reference.

Remaining Task

- **Edge Case:** Consider implementing error handling for edge cases such as invalid destination address.Provide informative error messages or prompts to guide users in correcting their input.
- **Refined Scope of Returned List:** Evaluate the possibility of refining the scope of the returned list. This could involve optimizing the criteria for selecting top restaurants or introducing additional filters based on user preferences.

- **Refine Code BM25 Query Efficiency**: Focus on optimizing the efficiency of the BM25 query process.
    - Our current BM25 ranking algorith parameters need to be fine tuned. Consider code refactoring to improve execution speed.
    - Explore alternative data structures for storing Inverse Document Frequency (IDF) values to enhance retrieval efficiency.
- **Testing Code:**Conduct comprehensive testing to ensure the functionality and robustness of the Python application.
- **Documentation on the Code:**Add clear and concise comments throughout the codebase, explaining the purpose of functions, classes, and significant code blocks.
- **Project Report:** Compile a detailed project report that outlines the methodology, objectives, and implemented solution. Provide insights into the design decisions, algorithms used and challenges faced.
- **Make Presentation:** According to the project instructions, prepare a visually engaging presentation that effectively communicates the project's goals, methodology, and outcomes.
- **Category-Based Recommendation (Stretch Goal):** As a stretch goal, consider implementing category-based recommendations. This could involve categorizing restaurants and providing users with options to narrow down their preferences based on specific categories (e.g., cuisine type, ambiance).

● Challenges/Issues

One of the primary challenges we encounter revolves around the enormity of the Yelp Open dataset (1) and optimizing the efficiency of our ranking algorithm (2). To address the first challenge, we've implemented a robust parsing method that preprocesses the dataset, significantly improving query speed. Regarding the second challenge, as highlighted earlier, fine-tuning our BM25 ranking algorithm parameters is imperative. This involves a code refactoring process aimed at enhancing execution speed and exploring alternative data structures for storing Inverse Document Frequency (IDF) values, thereby augmenting retrieval efficiency.

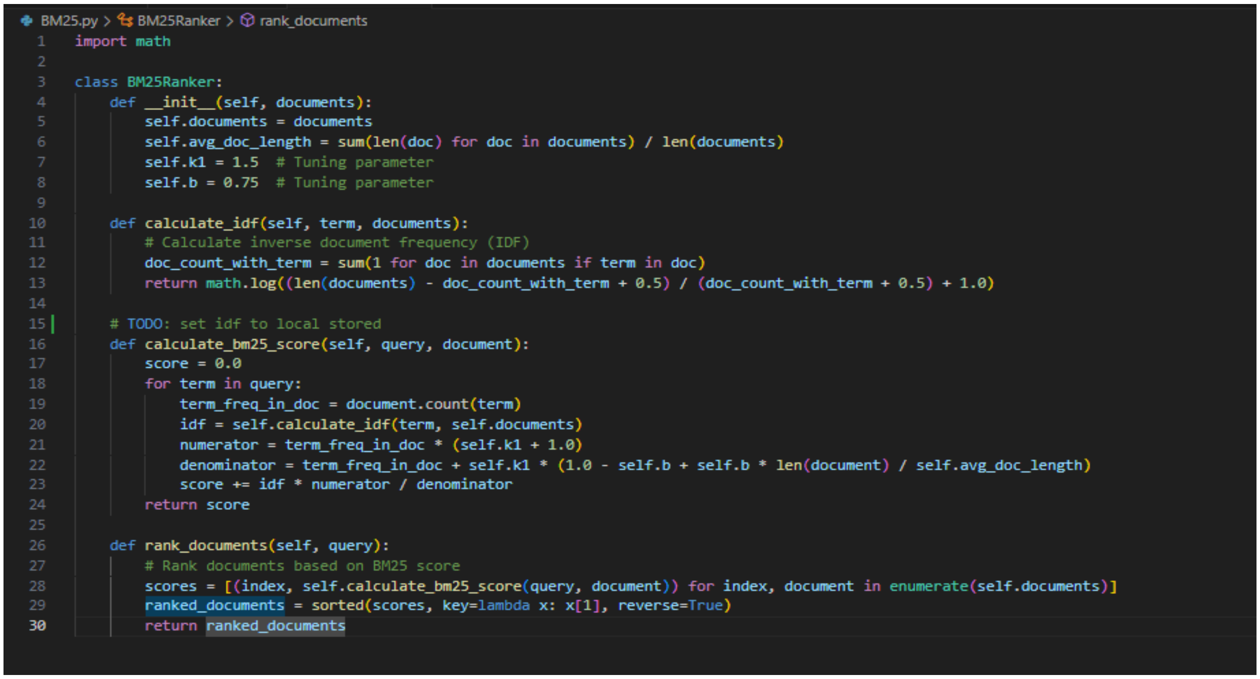
In addition to these technical challenges, a critical decision lies ahead in terms of project design. We must deliberate on whether to base the project solely on review scores, incorporate user preferences and keywords, or strike a balance between both approaches. Currently, our code implementation focuses solely on review scores.

However, to enrich our recommendation system, we are contemplating the enhancement of our algorithm to generate recommendations based on a combination of review scores and user preferences, if time permits.

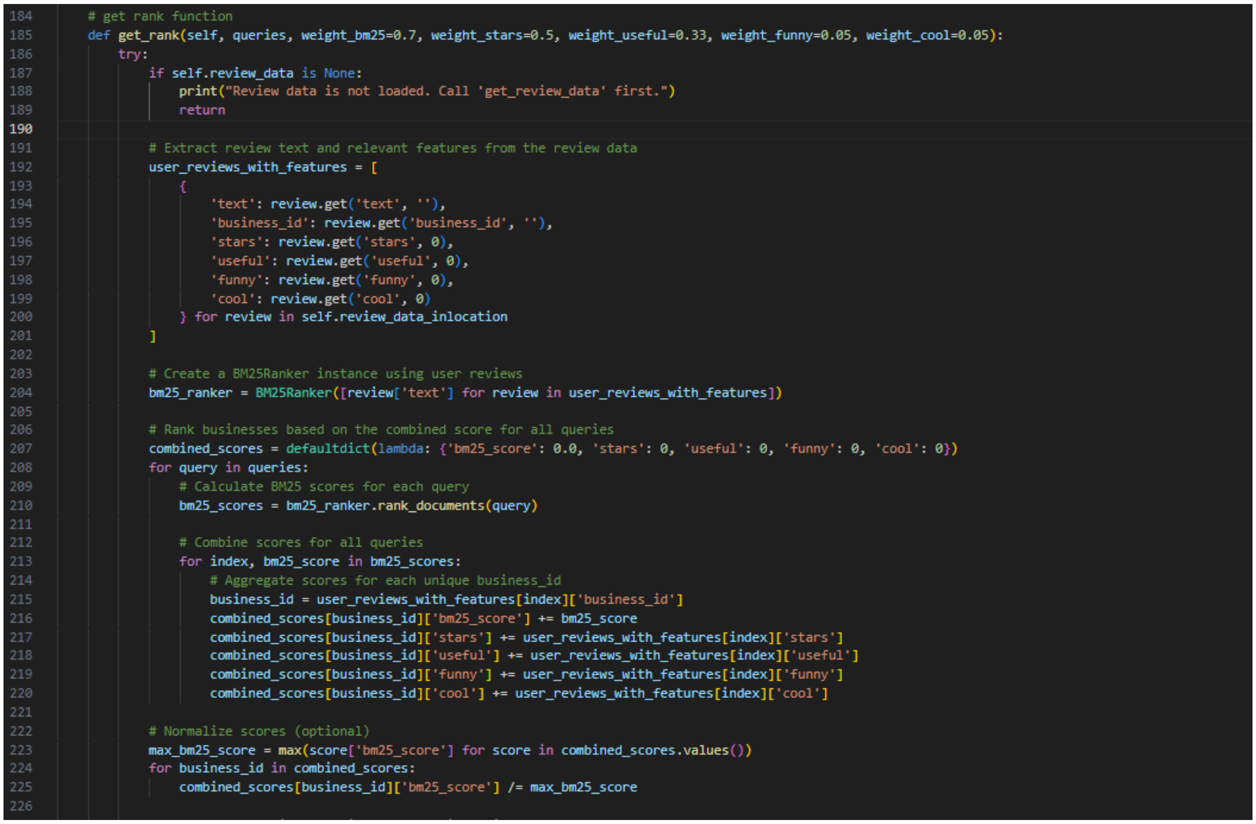
This strategic decision involves a nuanced understanding of user expectations and the overarching goals of the project. While prioritizing review scores provides a straightforward approach, integrating user preferences and keywords can offer a more personalized and context-aware recommendation system. Balancing both aspects could potentially yield a comprehensive and refined user experience, contributing to the overall success of the project. However, given the limited time left for this task, we consider this as a stretch goal to implement if we have time after completing the MVP.

In conclusion, our ongoing efforts involve not only technical optimizations for handling large datasets and refining algorithmic efficiency but also a thoughtful consideration of the project's design philosophy, with the potential for a more sophisticated recommendation system that encompasses both review scores and user preferences.

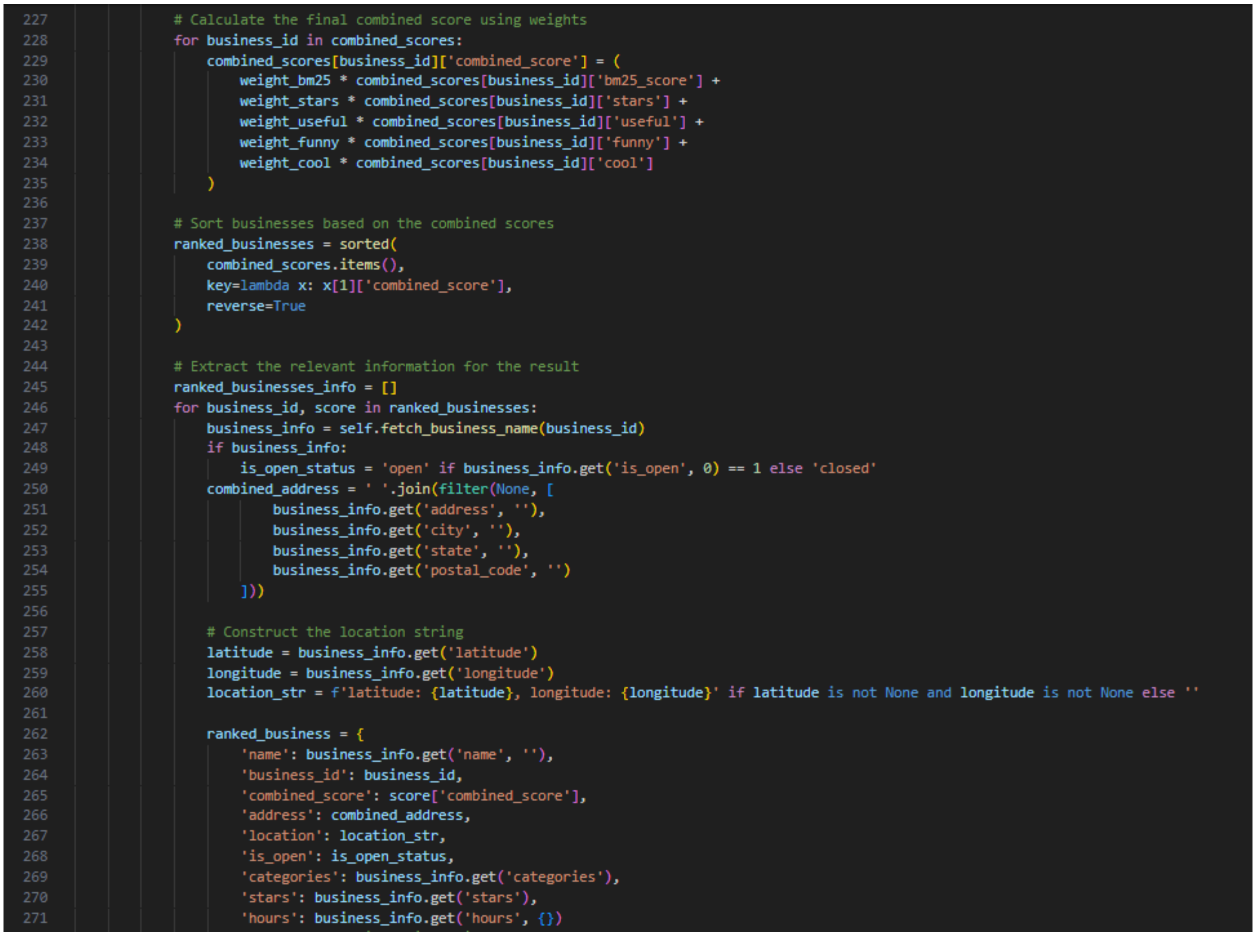
Code Snippets



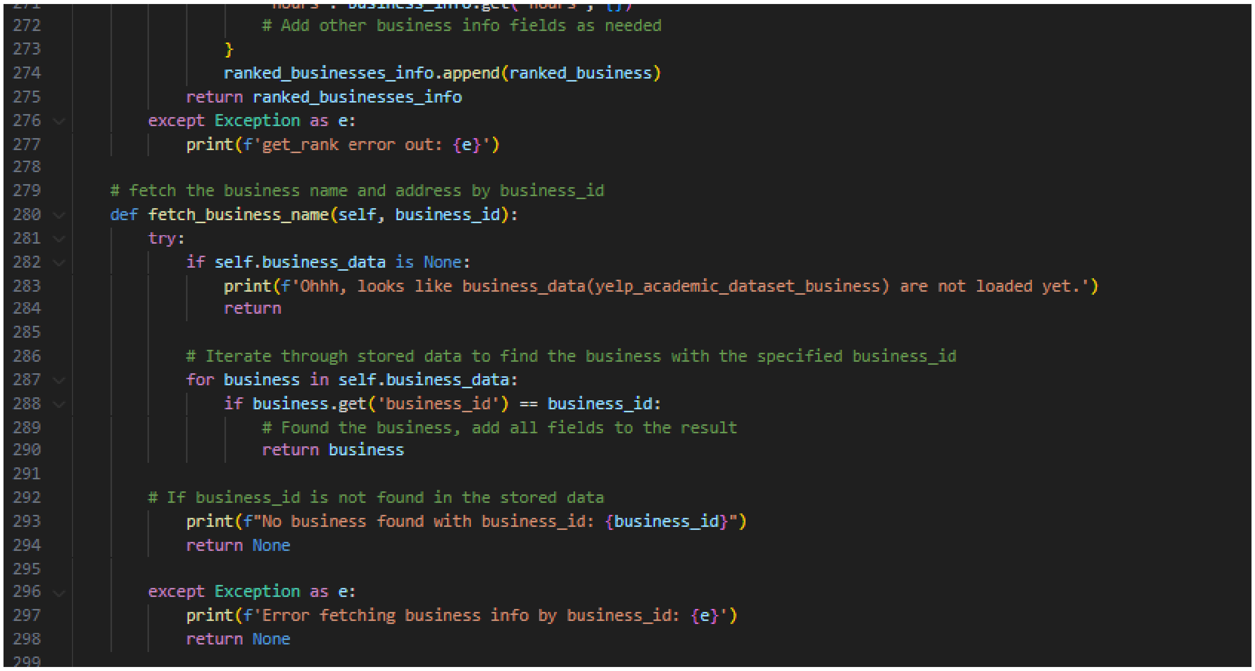
```python
import math

class BM25Ranker:
    def __init__(self, documents):
        self.documents = documents
        self.avg_doc_length = sum(len(doc) for doc in documents) / len(documents)
        self.k1 = 1.5  # Tuning parameter
        self.b = 0.75  # Tuning parameter

    def calculate_idf(self, term, documents):
        # Calculate inverse document frequency (IDF)
        doc_count_with_term = sum(1 for doc in documents if term in doc)
        return math.log((len(documents) - doc_count_with_term + 0.5) / (doc_count_with_term + 0.5) + 1.0)

    # TODO: set idf to local stored
    def calculate_bm25_score(self, query, document):
        score = 0.0
        for term in query:
            term_freq_in_doc = document.count(term)
            idf = self.calculate_idf(term, self.documents)
            numerator = term_freq_in_doc * (self.k1 + 1.0)
            denominator = term_freq_in_doc + self.k1 * (1.0 - self.b + self.b * len(document) / self.avg_doc_length)
            score += idf * numerator / denominator
        return score

    def rank_documents(self, query):
        # Rank documents based on BM25 score
        scores = [(index, self.calculate_bm25_score(query, document)) for index, document in enumerate(self.documents)]
        ranked_documents = sorted(scores, key=lambda x: x[1], reverse=True)
        return ranked_documents
```



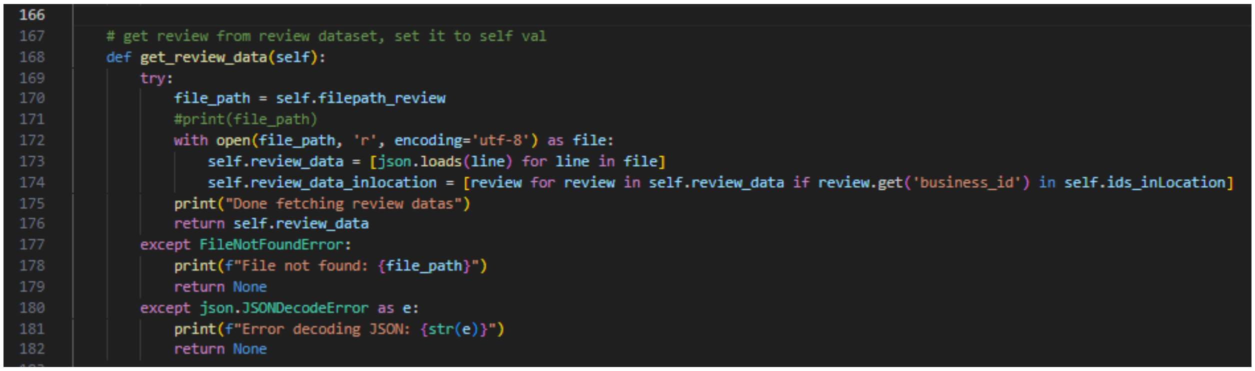
```python
    # get rank function
    def get_rank(self, queries, weight_bm25=0.7, weight_stars=0.5, weight_useful=0.33, weight_funny=0.05, weight_cool=0.05):
        try:
            if self.review_data is None:
                print("Review data is not loaded. Call 'get_review_data' first.")
                return

            # Extract review text and relevant features from the review data
            user_reviews_with_features = [
                {
                    'text': review.get('text', ''),
                    'business_id': review.get('business_id', ''),
                    'stars': review.get('stars', 0),
                    'useful': review.get('useful', 0),
                    'funny': review.get('funny', 0),
                    'cool': review.get('cool', 0)
                } for review in self.review_data_inlocation
            ]

            # Create a BM25Ranker instance using user reviews
            bm25_ranker = BM25Ranker([review['text'] for review in user_reviews_with_features])

            # Rank businesses based on the combined score for all queries
            combined_scores = defaultdict(lambda: {'bm25_score': 0.0, 'stars': 0, 'useful': 0, 'funny': 0, 'cool': 0})
            for query in queries:
                # Calculate BM25 scores for each query
                bm25_scores = bm25_ranker.rank_documents(query)

                # Combine scores for all queries
                for index, bm25_score in bm25_scores:
                    # Aggregate scores for each unique business_id
                    business_id = user_reviews_with_features[index]['business_id']
                    combined_scores[business_id]['bm25_score'] += bm25_score
                    combined_scores[business_id]['stars'] += user_reviews_with_features[index]['stars']
                    combined_scores[business_id]['useful'] += user_reviews_with_features[index]['useful']
                    combined_scores[business_id]['funny'] += user_reviews_with_features[index]['funny']
                    combined_scores[business_id]['cool'] += user_reviews_with_features[index]['cool']

            # Normalize scores (optional)
            max_bm25_score = max(score['bm25_score'] for score in combined_scores.values())
            for business_id in combined_scores:
                combined_scores[business_id]['bm25_score'] /= max_bm25_score

```



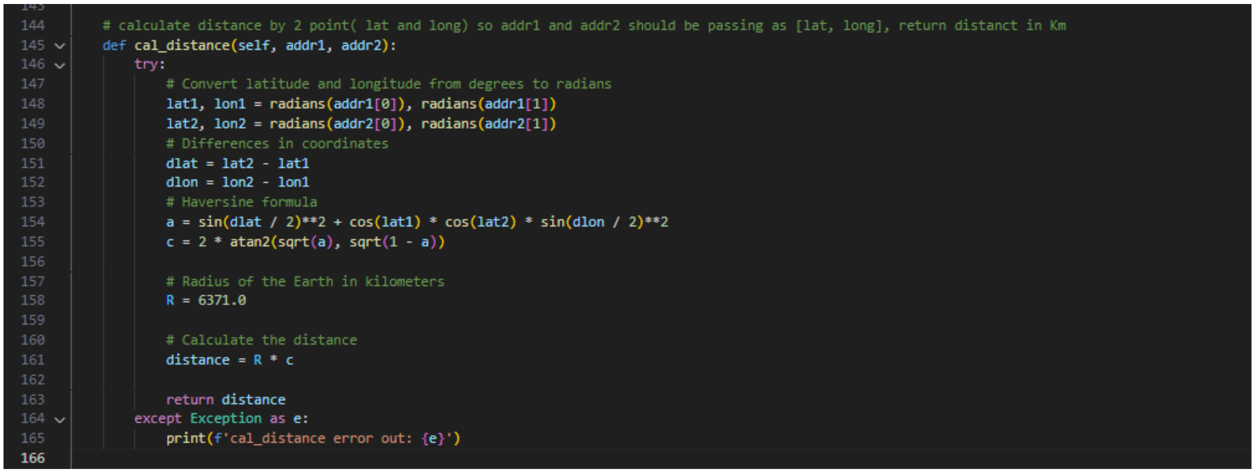
```python
            # Calculate the final combined score using weights
            for business_id in combined_scores:
                combined_scores[business_id]['combined_score'] = (
                    weight_bm25 * combined_scores[business_id]['bm25_score'] +
                    weight_stars * combined_scores[business_id]['stars'] +
                    weight_useful * combined_scores[business_id]['useful'] +
                    weight_funny * combined_scores[business_id]['funny'] +
                    weight_cool * combined_scores[business_id]['cool']
                )

            # Sort businesses based on the combined scores
            ranked_businesses = sorted(
                combined_scores.items(),
                key=lambda x: x[1]['combined_score'],
                reverse=True
            )

            # Extract the relevant information for the result
            ranked_businesses_info = []
            for business_id, score in ranked_businesses:
                business_info = self.fetch_business_name(business_id)
                if business_info:
                    is_open_status = 'open' if business_info.get('is_open', 0) == 1 else 'closed'
                combined_address = ' '.join(filter(None, [
                        business_info.get('address', ''),
                        business_info.get('city', ''),
                        business_info.get('state', ''),
                        business_info.get('postal_code', '')
                    ]))

                # Construct the location string
                latitude = business_info.get('latitude')
                longitude = business_info.get('longitude')
                location_str = f'latitude: {latitude}, longitude: {longitude}' if latitude is not None and longitude is not None else ''

                ranked_business = {
                    'name': business_info.get('name', ''),
                    'business_id': business_id,
                    'combined_score': score['combined_score'],
                    'address': combined_address,
                    'location': location_str,
                    'is_open': is_open_status,
                    'categories': business_info.get('categories'),
                    'stars': business_info.get('stars'),
                    'hours': business_info.get('hours', {})
```



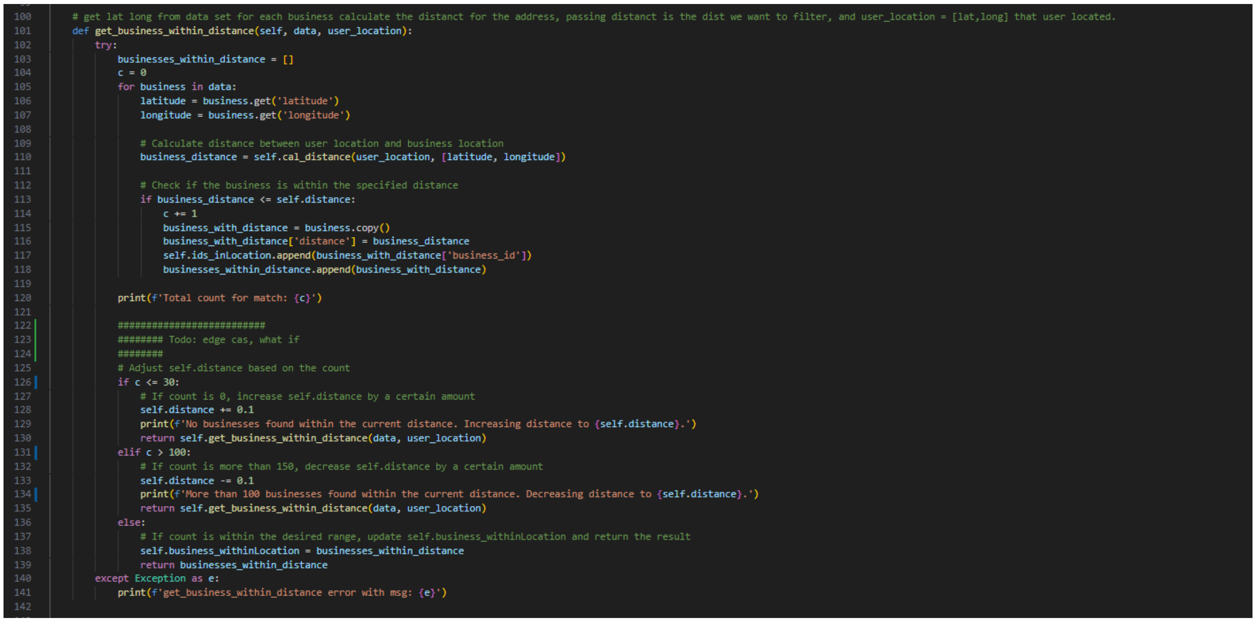
```
                    'hours': business_info.get('hours', [])
                    # Add other business info fields as needed
                }
                ranked_businesses_info.append(ranked_business)
            return ranked_businesses_info
        except Exception as e:
            print(f'get_rank error out: {e}')

    # fetch the business name and address by business_id
    def fetch_business_name(self, business_id):
        try:
            if self.business_data is None:
                print(f'Ohhh, looks like business_data(yelp_academic_dataset_business) are not loaded yet.')
                return

            # Iterate through stored data to find the business with the specified business_id
            for business in self.business_data:
                if business.get('business_id') == business_id:
                    # Found the business, add all fields to the result
                    return business

        # If business_id is not found in the stored data
            print(f"No business found with business_id: {business_id}")
            return None

        except Exception as e:
            print(f'Error fetching business info by business_id: {e}')
            return None
```



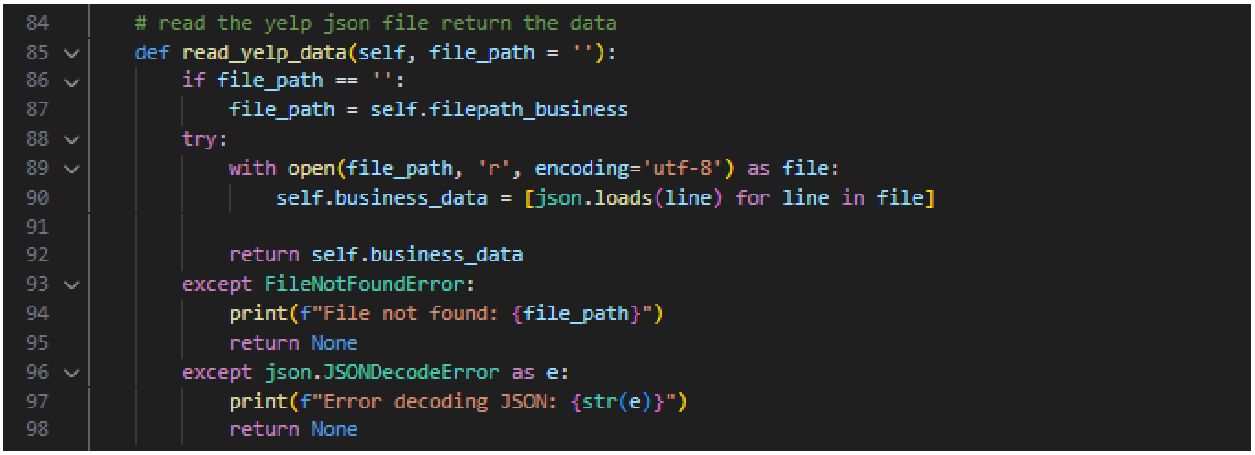
```
    # get review from review dataset, set it to self val
    def get_review_data(self):
        try:
            file_path = self.filepath_review
            #print(file_path)
            with open(file_path, 'r', encoding='utf-8') as file:
                self.review_data = [json.loads(line) for line in file]
                self.review_data_inlocation = [review for review in self.review_data if review.get('business_id') in self.ids_inLocation]
            print("Done fetching review datas")
            return self.review_data
        except FileNotFoundError:
            print(f"File not found: {file_path}")
            return None
        except json.JSONDecodeError as e:
            print(f"Error decoding JSON: {str(e)}")
            return None
```



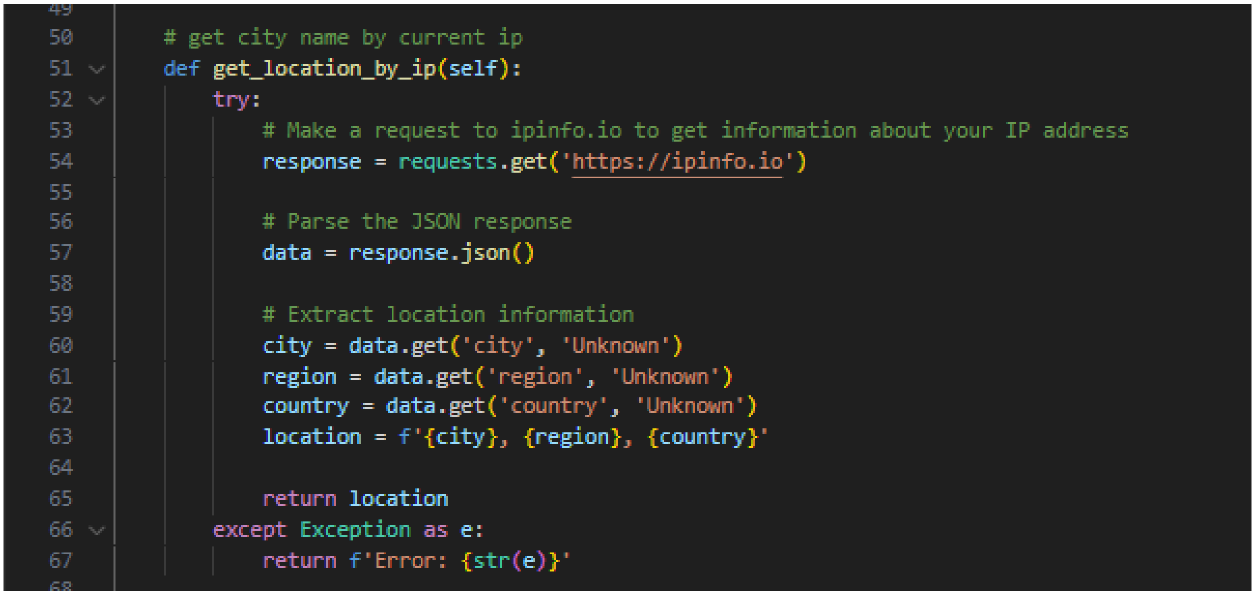
```
    # calculate distance by 2 point( lat and long) so addr1 and addr2 should be passing as [lat, long], return distanct in Km
    def cal_distance(self, addr1, addr2):
        try:
            # Convert latitude and longitude from degrees to radians
            lat1, lon1 = radians(addr1[0]), radians(addr1[1])
            lat2, lon2 = radians(addr2[0]), radians(addr2[1])
            # Differences in coordinates
            dlat = lat2 - lat1
            dlon = lon2 - lon1
            # Haversine formula
            a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
            c = 2 * atan2(sqrt(a), sqrt(1 - a))

            # Radius of the Earth in kilometers
            R = 6371.0

            # Calculate the distance
            distance = R * c

            return distance
        except Exception as e:
            print(f'cal_distance error out: {e}')
```



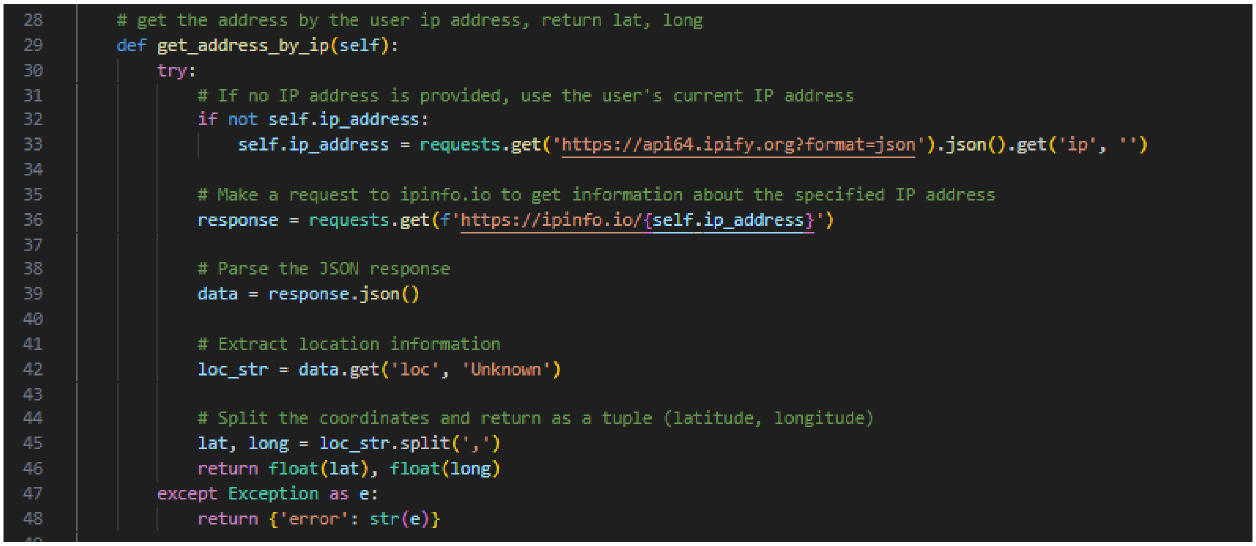
```python
    # get lat long from data set for each business calculate the distanct for the address, passing distanct is the dist we want to filter, and user_location = [lat,long] that user located.
    def get_business_within_distance(self, data, user_location):
        try:
            businesses_within_distance = []
            c = 0
            for business in data:
                latitude = business.get('latitude')
                longitude = business.get('longitude')

                # Calculate distance between user location and business location
                business_distance = self.cal_distance(user_location, [latitude, longitude])

                # Check if the business is within the specified distance
                if business_distance <= self.distance:
                    c += 1
                    business_with_distance = business.copy()
                    business_with_distance['distance'] = business_distance
                    self.ids_inLocation.append(business_with_distance['business_id'])
                    businesses_within_distance.append(business_with_distance)

            print(f'Total count for match: {c}')

            ##########################
            ######## Todo: edge cas, what if
            ########
            # Adjust self.distance based on the count
            if c <= 30:
                # If count is 0, increase self.distance by a certain amount
                self.distance += 0.1
                print(f'No businesses found within the current distance. Increasing distance to {self.distance}.')
                return self.get_business_within_distance(data, user_location)
            elif c > 100:
                # If count is more than 150, decrease self.distance by a certain amount
                self.distance -= 0.1
                print(f'More than 100 businesses found within the current distance. Decreasing distance to {self.distance}.')
                return self.get_business_within_distance(data, user_location)
            else:
                # If count is within the desired range, update self.business_withinLocation and return the result
                self.business_withinLocation = businesses_within_distance
                return businesses_within_distance
        except Exception as e:
            print(f'get_business_within_distance error with msg: {e}')

```



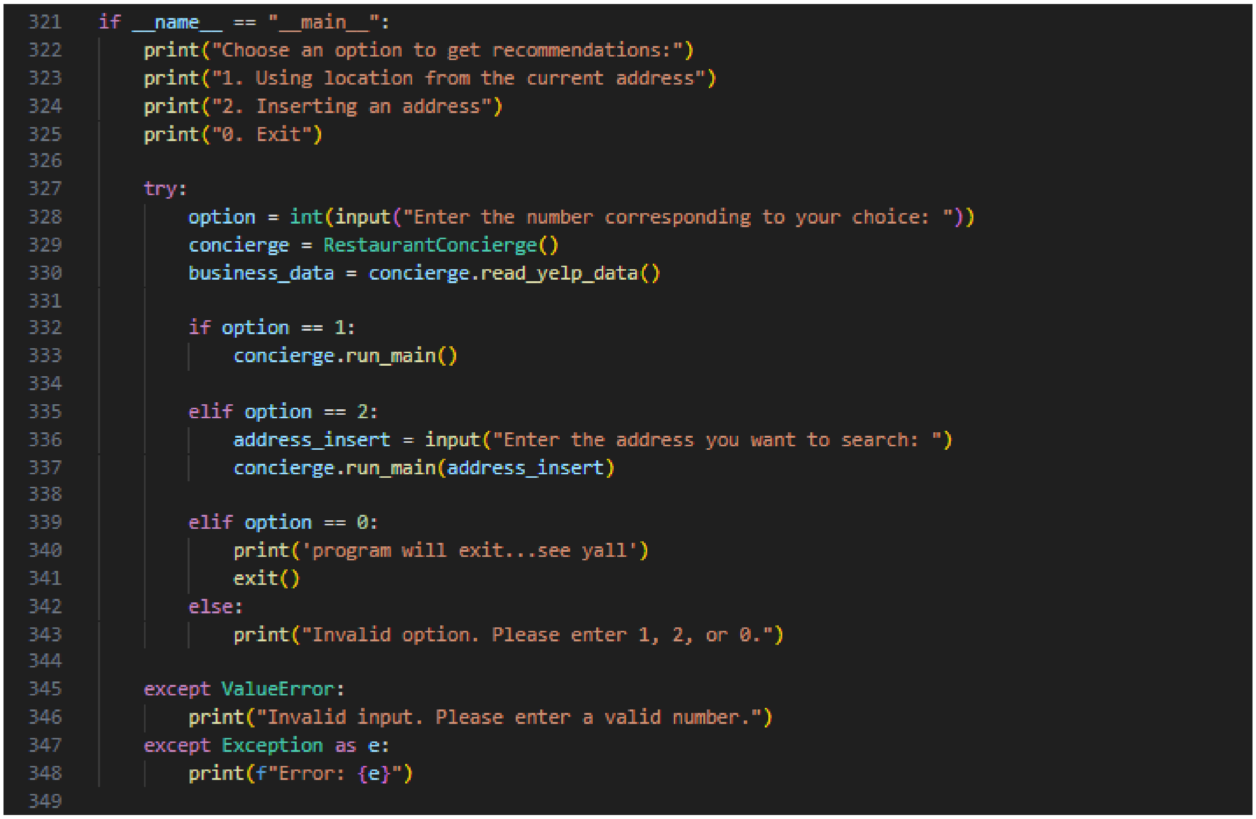
```python
    # read the yelp json file return the data
    def read_yelp_data(self, file_path = ''):
        if file_path == '':
            file_path = self.filepath_business
        try:
            with open(file_path, 'r', encoding='utf-8') as file:
                self.business_data = [json.loads(line) for line in file]

            return self.business_data
        except FileNotFoundError:
            print(f"File not found: {file_path}")
            return None
        except json.JSONDecodeError as e:
            print(f"Error decoding JSON: {str(e)}")
            return None
```



```python
    # get city name by current ip
    def get_location_by_ip(self):
        try:
            # Make a request to ipinfo.io to get information about your IP address
            response = requests.get('https://ipinfo.io')

            # Parse the JSON response
            data = response.json()

            # Extract location information
            city = data.get('city', 'Unknown')
            region = data.get('region', 'Unknown')
            country = data.get('country', 'Unknown')
            location = f'{city}, {region}, {country}'

            return location
        except Exception as e:
            return f'Error: {str(e)}'
```

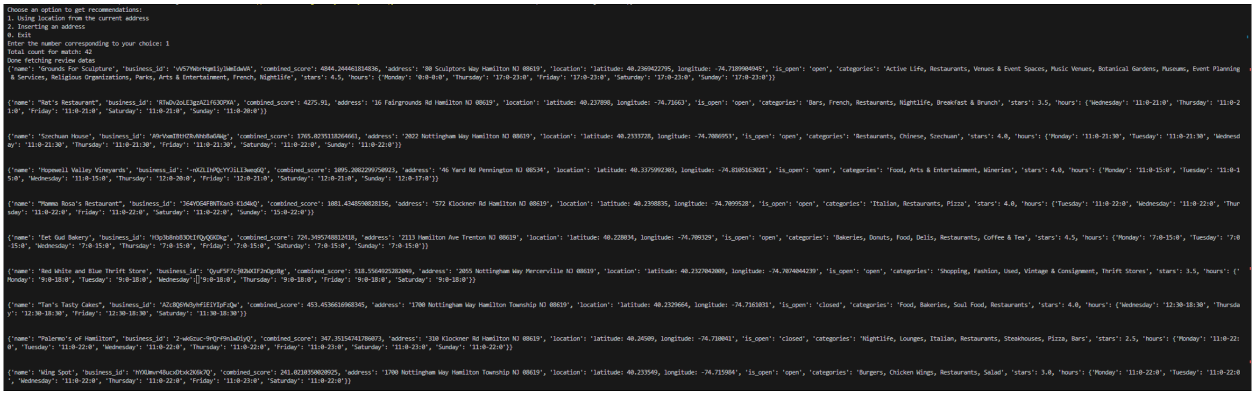


```python
    # get the address by the user ip address, return lat, long
    def get_address_by_ip(self):
        try:
            # If no IP address is provided, use the user's current IP address
            if not self.ip_address:
                self.ip_address = requests.get('https://api64.ipify.org?format=json').json().get('ip', '')

            # Make a request to ipinfo.io to get information about the specified IP address
            response = requests.get(f'https://ipinfo.io/{self.ip_address}')

            # Parse the JSON response
            data = response.json()

            # Extract location information
            loc_str = data.get('loc', 'Unknown')

            # Split the coordinates and return as a tuple (latitude, longitude)
            lat, long = loc_str.split(',')
            return float(lat), float(long)
        except Exception as e:
            return {'error': str(e)}
```



```python
if __name__ == "__main__":
    print("Choose an option to get recommendations:")
    print("1. Using location from the current address")
    print("2. Inserting an address")
    print("0. Exit")

    try:
        option = int(input("Enter the number corresponding to your choice: "))
        concierge = RestaurantConcierge()
        business_data = concierge.read_yelp_data()

        if option == 1:
            concierge.run_main()

        elif option == 2:
            address_insert = input("Enter the address you want to search: ")
            concierge.run_main(address_insert)

        elif option == 0:
            print('program will exit...see yall')
            exit()
        else:
            print("Invalid option. Please enter 1, 2, or 0.")

    except ValueError:
        print("Invalid input. Please enter a valid number.")
    except Exception as e:
        print(f"Error: {e}")
```

Code result:



```
Choose an option to get recommendations:
1. Using location from the current address
2. Inserting an address
0. Exit
Enter the number corresponding to your choice: 1
Total count for match: 42
Done fetching review datas
{'name': 'Grounds For Sculpture', 'business_id': 'vV57YWbrHqm1iylWmIdwVA', 'combined_score': 4844.244461814836, 'address': '80 Sculptors Way Hamilton NJ 08619', 'location': 'latitude: 40.2369422795, longitude: -74.7189904945', 'is_open': 'open', 'categories': 'Active Life, Restaurants, Venues & Event Spaces, Music Venues, Botanical Gardens, Museums, Event Planning
 & Services, Religious Organizations, Parks, Arts & Entertainment, French, Nightlife', 'stars': 4.5, 'hours': {'Monday': '0:0-0:0', 'Thursday': '17:0-23:0', 'Friday': '17:0-23:0', 'Saturday': '17:0-23:0', 'Sunday': '17:0-23:0'}}


{'name': "Rat's Restaurant", 'business_id': 'RTwDv2oLE3gzAZlf63OPXA', 'combined_score': 4275.91, 'address': '16 Fairgrounds Rd Hamilton NJ 08619', 'location': 'latitude: 40.237898, longitude: -74.71663', 'is_open': 'open', 'categories': 'Bars, French, Restaurants, Nightlife, Breakfast & Brunch', 'stars': 3.5, 'hours': {'Wednesday': '11:0-21:0', 'Thursday': '11:0-2
1:0', 'Friday': '11:0-21:0', 'Saturday': '11:0-21:0', 'Sunday': '11:0-20:0'}}


{'name': 'Szechuan House', 'business_id': 'A9rVxmIBtHZRvNhbBaGAWg', 'combined_score': 1765.0235118264661, 'address': '2022 Nottingham Way Hamilton NJ 08619', 'location': 'latitude: 40.2333728, longitude: -74.7086953', 'is_open': 'open', 'categories': 'Restaurants, Chinese, Szechuan', 'stars': 4.0, 'hours': {'Monday': '11:0-21:30', 'Tuesday': '11:0-21:30', 'Wednesd
ay': '11:0-21:30', 'Thursday': '11:0-21:30', 'Friday': '11:0-21:30', 'Saturday': '11:0-22:0', 'Sunday': '11:0-22:0'}}


{'name': 'Hopewell Valley Vineyards', 'business_id': '-nXZLIhPQcYYJiLI3weqGQ', 'combined_score': 1095.2082299750923, 'address': '46 Yard Rd Pennington NJ 08534', 'location': 'latitude: 40.3375992303, longitude: -74.8105163021', 'is_open': 'open', 'categories': 'Food, Arts & Entertainment, Wineries', 'stars': 4.0, 'hours': {'Monday': '11:0-15:0', 'Tuesday': '11:0-1
5:0', 'Wednesday': '11:0-15:0', 'Thursday': '12:0-20:0', 'Friday': '12:0-21:0', 'Saturday': '12:0-21:0', 'Sunday': '12:0-17:0'}}


{'name': "Mamma Rosa's Restaurant", 'business_id': 'J64YOG4FBNTKan3-K1d4kQ', 'combined_score': 1081.4348590828156, 'address': '572 Klockner Rd Hamilton NJ 08619', 'location': 'latitude: 40.2398835, longitude: -74.7099528', 'is_open': 'open', 'categories': 'Italian, Restaurants, Pizza', 'stars': 4.0, 'hours': {'Tuesday': '11:0-22:0', 'Wednesday': '11:0-22:0', 'Thur
sday': '11:0-22:0', 'Friday': '11:0-22:0', 'Saturday': '11:0-22:0', 'Sunday': '15:0-22:0'}}


{'name': 'Eet Gud Bakery', 'business_id': 'H3p3b8nbB3OtIfQyQGKDkg', 'combined_score': 724.3495748812418, 'address': '2113 Hamilton Ave Trenton NJ 08619', 'location': 'latitude: 40.228034, longitude: -74.709329', 'is_open': 'open', 'categories': 'Bakeries, Donuts, Food, Delis, Restaurants, Coffee & Tea', 'stars': 4.5, 'hours': {'Monday': '7:0-15:0', 'Tuesday': '7:0
-15:0', 'Wednesday': '7:0-15:0', 'Thursday': '7:0-15:0', 'Friday': '7:0-15:0', 'Saturday': '7:0-15:0', 'Sunday': '7:0-15:0'}}


{'name': 'Red White and Blue Thrift Store', 'business_id': 'QyuF5F7cj02WXIF2nOgzBg', 'combined_score': 518.5564925282049, 'address': '2055 Nottingham Way Mercerville NJ 08619', 'location': 'latitude: 40.2327042009, longitude: -74.7074044239', 'is_open': 'open', 'categories': 'Shopping, Fashion, Used, Vintage & Consignment, Thrift Stores', 'stars': 3.5, 'hours': {'
Monday': '9:0-18:0', 'Tuesday': '9:0-18:0', 'Wednesday': '9:0-18:0', 'Thursday': '9:0-18:0', 'Friday': '9:0-18:0', 'Saturday': '9:0-18:0'}}


{'name': "Tan's Tasty Cakes", 'business_id': 'AZc8Q6YW3yhfiEiYIpFzQw', 'combined_score': 453.4536616968345, 'address': '1700 Nottingham Way Hamilton Township NJ 08619', 'location': 'latitude: 40.2329664, longitude: -74.7161031', 'is_open': 'closed', 'categories': 'Food, Bakeries, Soul Food, Restaurants', 'stars': 4.0, 'hours': {'Wednesday': '12:30-18:30', 'Thursda
y': '12:30-18:30', 'Friday': '12:30-18:30', 'Saturday': '11:30-18:30'}}


{'name': "Palermo's of Hamilton", 'business_id': '2-wkGzuc-9rQrf9nlwDiyQ', 'combined_score': 347.35154741786073, 'address': '310 Klockner Rd Hamilton NJ 08619', 'location': 'latitude: 40.24509, longitude: -74.710041', 'is_open': 'closed', 'categories': 'Nightlife, Lounges, Italian, Restaurants, Steakhouses, Pizza, Bars', 'stars': 2.5, 'hours': {'Monday': '11:0-22:
0', 'Tuesday': '11:0-22:0', 'Wednesday': '11:0-22:0', 'Thursday': '11:0-22:0', 'Friday': '11:0-23:0', 'Saturday': '11:0-23:0', 'Sunday': '11:0-22:0'}}


{'name': 'Wing Spot', 'business_id': 'hYXUmvr48ucxDtxk2K6k7Q', 'combined_score': 241.0210350020925, 'address': '1700 Nottingham Way Hamilton Township NJ 08619', 'location': 'latitude: 40.233549, longitude: -74.715984', 'is_open': 'open', 'categories': 'Burgers, Chicken Wings, Restaurants, Salad', 'stars': 3.0, 'hours': {'Monday': '11:0-22:0', 'Tuesday': '11:0-22:0
', 'Wednesday': '11:0-22:0', 'Thursday': '11:0-22:0', 'Friday': '11:0-23:0', 'Saturday': '11:0-22:0'}}
```