Data Checkpoint

Name: Kedong He
Uniqname: kedongh
Github repo: https://github.com/kedongh/507_final_proj

Data Source

My data comes from two sources. One is Web API to search some business related to the condition, the other one is Yelp business websites to scrape detailed information.

Firstly, I use Web API to fetch basic information of business according to the given term and the location and serialize the response to JSON format. The endpoint is https://api.yelp.com/v3/businesses/search, and there are two parameters for the user to input. The first parameter is search term. It can be a general description, like “food” and “restaurant”, or even a detailed business name, like “Starbucks”. The second parameter is location. It indicates the geographic area to be used when searching for businesses. Each response would contain at most 50 business results. In the response body, some parameters are of interest. I select rating, price, business name, address, Yelp url, review count, phone number, city and Yelp id as the basic information to present to the user and save to the database. To be time efficient, I use caching to store the retrieved results after the first time search. The following is the snapshot of the code that implements caching.

```
def make_request(baseurl, params):
    '''Make a request to the Web API using the baseurl and params

    Args;
        baseurl (str): The URL for the API endpoint.
        params (dict): A dictionary of param:value pairs.

    Returns:
        results (dict): The JSON response from the request.
    '''
    response = requests.get(baseurl, params=params, headers=headers)
    results = response.json()
    return results


def make_request_with_cache(baseurl, term='', location='', count=50):
    ''' Check the cache for a saved result for this baseurl+params:values
    combo. If the result is found, return it. Otherwise send a new
    request, save it, then return it.

    Args:
        baseurl (str): The URL for the API endpoint
        term (str): The search term passes to the API.
        location (str): The search location passes to the API.
        count (int): The number of business results to return.

    Return:
        results (dict): The JSON response from the request.
    '''
    params = {
        'term': term.lower().replace(" ", "+"),
        'location': location.lower().replace(" ", "+"),
        'limit': count
    }
    request_key = construct_unique_key(baseurl=baseurl, params=params)

    if request_key in CACHE_DICT:
        # The data has been fetched before and stored in the cache
        return CACHE_DICT[request_key]
    else:
        results = make_request(baseurl=baseurl, params=params)
        CACHE_DICT[request_key] = results
        save_cache(cache_dict=CACHE_DICT)
        return results
```

Figure 1: Snapshot of code implementing caching.

The second source is Yelp business website. I plan to scrape some pictures and recent reviews loaded by customers with Beautiful Soup. To scrape a page, I will first get the HTML page contents in text format and then “parse” the contents, looking for particular tags with particular attributes and then extracting the

contents of those tags. In each Yelp business website, I scrape at most 3 pictures and at most 3 top and recent reviews. Since the pictures and reviews are likely to be updated day-by-day, although it is possible to use caching, the information stored in the cache might be out-of-date. So I decide not to use caching for this data source.

Database

I create two databases. One, named as "business_info", is to store all of infor of business and the other one, named as "category_info", is to store some categories of business. The left picture is the database schema of "business_info" and the right picture is the schema of "category_info".

```
CREATE TABLE "business_info" (
    "yelp_id"    TEXT UNIQUE,
    "business_name" TEXT NOT NULL,
    "city"   TEXT NOT NULL,
    "phone_number"   TEXT NOT NULL,
    "review_count"   INTEGER NOT NULL,
    "rating"     REAL NOT NULL,
    "price" INTEGER NOT NULL,
    "address"    TEXT NOT NULL,
    "url"    TEXT NOT NULL,
    PRIMARY KEY("yelp_id")
);
```

```
CREATE TABLE "category_info" (
    "yelp_id"    TEXT UNIQUE,
    "category_1"     TEXT NOT NULL,
    "category_2"     TEXT NOT NULL,
    "category_3"     TEXT NOT NULL,
    PRIMARY KEY("yelp_id")
);
```

Figure 2: Database schema of "business_info" and "category_info".

It is worth noting that "yelp_id" is the primary key in two tables and the foreign key in "business_info" pointing to the business in "category_info". And the following two figures are snapshots of two tables.

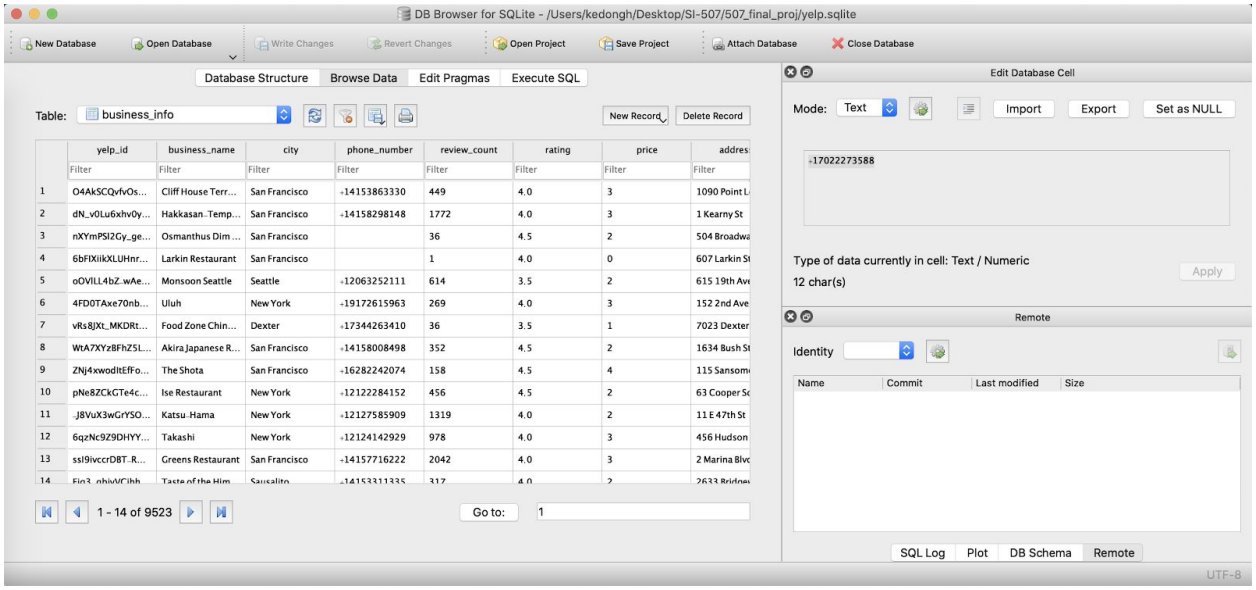

Figure 3: Snapshot of "business_info" table.

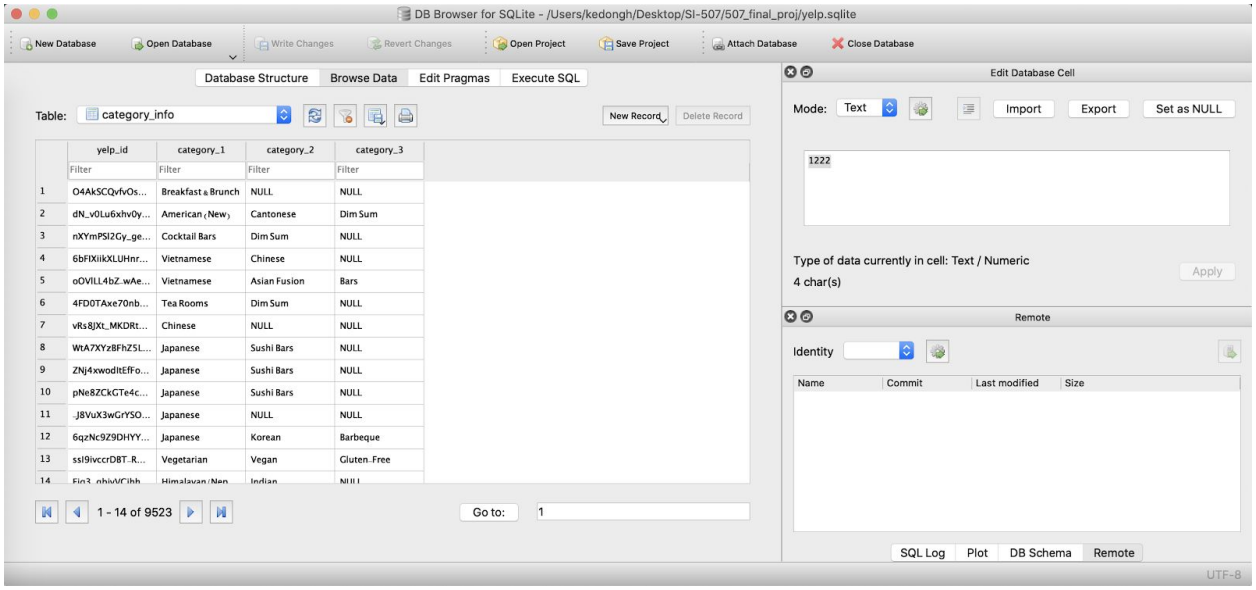

Figure 4: Snapshot of "category_info" table.

Interaction and Presentation Plans

I plan to use Flask platform to realize interaction and presentation. As shown in figure 5 (just a demo), there are two options. The first one "search" is to allow the user to get some business information after typing term and location. The second one is "summary" which is designed to give a rating/price/review_count summary for different business types or different cities. For better presentation, I decide to use Ploty to draw barplot or other formats (not decided yet).

Figure 5: Homepage.