Final Project Data CheckPoint

Project code

https://github.com/Huadous/final-project

Data sources

Categories information

1. **Origin** : Documentation Download


```
[
    {
        "alias": "3dprinting",
        "title": "3D Printing",
        "parents": [
            "localservices"
        ]
    },
    {
        "alias": "abruzzese",
        "title": "Abruzzese",
        "parents": [
            "italian"
        ],
        "country_whitelist": [
            "IT"
        ]
    },
    {
        "alias": "absinthebars",
        "title": "Absinthe Bars",
        "parents": [
            "bars"
        ],
        "country_whitelist": [
            "CZ"
        ]
    },
```


2. **Format** : JSON(> 1000 records)
3. **Data access and caching** : downloaded directly without additional verification methods, I used cache.
4. **Summary of data** : [≈ 1500 I will use 192]

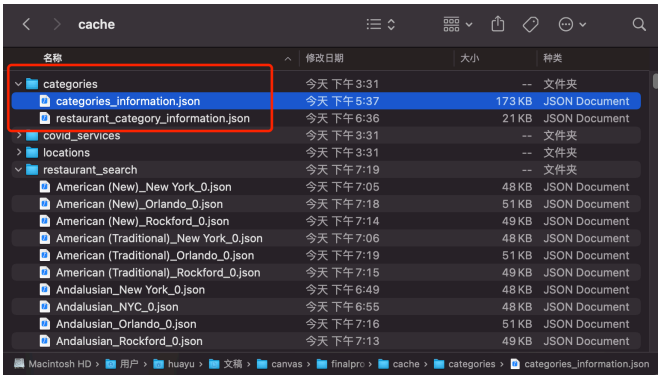
   It contains information about categories and available countries. Therefore, this data can be used as a benchmark for restaurant category search. Because this is an all-category file, which contains not only the categories of restaurants. What I need to do is to filter out the category of the restaurant from all categories.

   Important fields:

   - *"alias"* : alias of the child category, offer a different name for title.
   - *"title"* : title of the child category and will be used to find different type of restaurant.
   - *"parents"* : belongs to what parent category. The program use this to find which category belongs to restaurants.
   - *"country_whitelist"* : available countries (without this field means TO ALL THE COUNTRIES)
5. **Evidence of caching**

```
file_name = 'categories_information'
file_type = 'categories'
categories_information = cache.sync_cache(file_name, file_type)
if categories_information == {} or len(categories_information) == 0:
    categories_information_url_json = 'https://www.yelp.com/developers/documentation/v3/all_
    r = requests.get(categories_information_url_json)
    categories_information =json.loads(r.text)
    cache.save_cache(file_name, file_type, categories_information)

file_name = 'restaurant_category_information'
file_type = 'categories'
restaurant_category_information = cache.sync_cache(file_name, file_type)
if restaurant_category_information == {} or len(restaurant_category_information) == 0:
    restaurant_category_information = []
    for ele in categories_information:
        if 'parents' in ele and len(ele['parents']) > 0 and ele['parents'][0] == 'restaurant
            restaurant_category_information.append(ele)
cache.save_cache(file_name, file_type, restaurant_category_information)
```



ISO 3166-1 alpha-2 code

1. **Origin** : Documentation Download

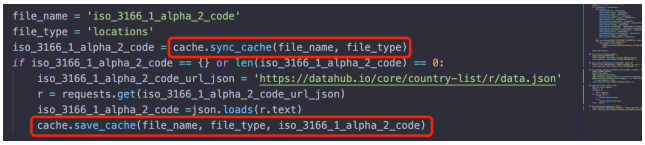

```
[{"Code": "AF", "Name": "Afghanistan"},{"Code": "AX", "Name": "\u00c5land Islands"},{"Code": "AL", "Name": "Albania"},{"Code": "DZ", "Name":
"Algeria"},{"Code": "AS", "Name": "American Samoa"},{"Code": "AD", "Name": "Andorra"},{"Code": "AO", "Name": "Angola"},{"Code": "AI", "Name":
"Anguilla"},{"Code": "AQ", "Name": "Antarctica"},{"Code": "AG", "Name": "Antigua and Barbuda"},{"Code": "AR", "Name": "Argentina"},{"Code": "AM",
"Name": "Armenia"},{"Code": "AW", "Name": "Aruba"},{"Code": "AU", "Name": "Australia"},{"Code": "AT", "Name": "Austria"},{"Code": "AZ", "Name":
"Azerbaijan"},{"Code": "BS", "Name": "Bahamas"},{"Code": "BH", "Name": "Bahrain"},{"Code": "BD", "Name": "Bangladesh"},{"Code": "BB", "Name":
"Barbados"},{"Code": "BY", "Name": "Belarus"},{"Code": "BE", "Name": "Belgium"},{"Code": "BZ", "Name": "Belize"},{"Code": "BJ", "Name": "Benin"},
{"Code": "BM", "Name": "Bermuda"},{"Code": "BT", "Name": "Bhutan"},{"Code": "BO", "Name": "Bolivia, Plurinational State of"},{"Code": "BQ", "Name":
"Bonaire, Sint Eustatius and Saba"},{"Code": "BA", "Name": "Bosnia and Herzegovina"},{"Code": "BW", "Name": "Botswana"},{"Code": "BV", "Name":
"Bouvet Island"},{"Code": "BR", "Name": "Brazil"},{"Code": "IO", "Name": "British Indian Ocean Territory"},{"Code": "BN", "Name": "Brunei
Darussalam"},{"Code": "BG", "Name": "Bulgaria"},{"Code": "BF", "Name": "Burkina Faso"},{"Code": "BI", "Name": "Burundi"},{"Code": "KH", "Name":
"Cambodia"},{"Code": "CM", "Name": "Cameroon"},{"Code": "CA", "Name": "Canada"},{"Code": "CV", "Name": "Cape Verde"},{"Code": "KY", "Name": "Cayman
Islands"},{"Code": "CF", "Name": "Central African Republic"},{"Code": "TD", "Name": "Chad"},{"Code": "CL", "Name": "Chile"},{"Code": "CN", "Name":
"China"},{"Code": "CX", "Name": "Christmas Island"},{"Code": "CC", "Name": "Cocos (Keeling) Islands"},{"Code": "CO", "Name": "Colombia"},{"Code":
"KM", "Name": "Comoros"},{"Code": "CG", "Name": "Congo"},{"Code": "CD", "Name": "Congo, the Democratic Republic of the"},{"Code": "CK", "Name":
"Cook Islands"},{"Code": "CR", "Name": "Costa Rica"},{"Code": "CI", "Name": "C\u00f4te d'Ivoire"},{"Code": "HR", "Name": "Croatia"},{"Code": "CU",
"Name": "Cuba"},{"Code": "CW", "Name": "Cura\u00e7ao"},{"Code": "CY", "Name": "Cyprus"},{"Code": "CZ", "Name": "Czech Republic"},{"Code": "DK",
"Name": "Denmark"},{"Code": "DJ", "Name": "Djibouti"},{"Code": "DM", "Name": "Dominica"},{"Code": "DO", "Name": "Dominican Republic"},{"Code":
"EC", "Name": "Ecuador"},{"Code": "EG", "Name": "Egypt"},{"Code": "SV", "Name": "El Salvador"},{"Code": "GQ", "Name": "Equatorial Guinea"},{"Code":
"ER", "Name": "Eritrea"},{"Code": "EE", "Name": "Estonia"},{"Code": "ET", "Name": "Ethiopia"},{"Code": "FK", "Name": "Falkland Islands
(Malvinas)"},{"Code": "FO", "Name": "Faroe Islands"},{"Code": "FJ", "Name": "Fiji"},{"Code": "FI", "Name": "Finland"},{"Code": "FR", "Name":
"France"},{"Code": "GF", "Name": "French Guiana"},{"Code": "PF", "Name": "French Polynesia"},{"Code": "TF", "Name": "French Southern Territories"},
{"Code": "GA", "Name": "Gabon"},{"Code": "GM", "Name": "Gambia"},{"Code": "GE", "Name": "Georgia"},{"Code": "DE", "Name": "Germany"},{"Code": "GH",
"Name": "Ghana"},{"Code": "GI", "Name": "Gibraltar"},{"Code": "GR", "Name": "Greece"},{"Code": "GL", "Name": "Greenland"},{"Code": "GD", "Name":
"Grenada"},{"Code": "GP", "Name": "Guadeloupe"},{"Code": "GU", "Name": "Guam"},{"Code": "GT", "Name": "Guatemala"},{"Code": "GG", "Name":
```

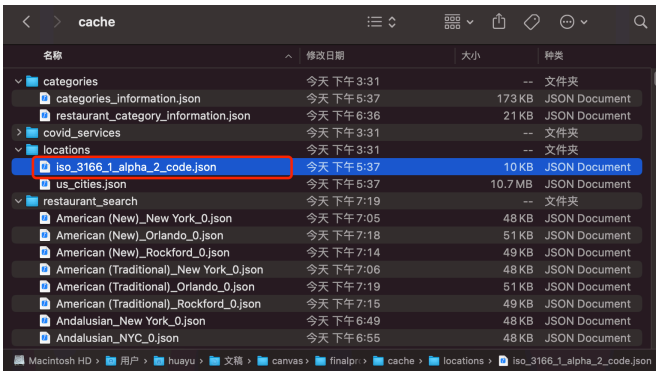

2. **Format** : JSON(< 1000 records)
3. **Data access and caching** : downloaded directly without additional verification methods, I used cache.
4. **Summary of data** : [≈ 250 I will use all of them]

   Because in the previous category file, there will be information about different restaurant categories in which countries provide search services. Therefore, it is necessary to use the abbreviations of the names of each country in this file to determine whether the restaurant in this category can be searched in the United States

   Important fields :

   - *"Code"* : code of the country, this field is the same as the *"country_whitelist"* and improve its readability by providing the full name of the country.
   - *"Name"* : name of the country, which is better for human reading.
5. **Evidence of caching**





United States Cities Database

1. **Origin**: Documentation Download

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	city	city_ascii	state_id	state_name	county_fips	county_nam	lat	lng	population	density	source	military	incorporate	timezone	ranking	zips	id
2	New York	New York	NY	New York	36061	New York	40.6943	-73.9249	18713220	10715	polygon	FALSE	TRUE	America/Ne	1	11229 11226	1840034016
3	Los Angeles	Los Angeles	CA	California	6037	Los Angeles	34.1139	-118.4068	12750807	3276	polygon	FALSE	TRUE	America/Los	1	90291 90293	1840020491
4	Chicago	Chicago	IL	Illinois	17031	Cook	41.8373	-87.6862	8604203	4574	polygon	FALSE	TRUE	America/Ch	1	60018 60649	1840000494
5	Miami	Miami	FL	Florida	12086	Miami-Dade	25.7839	-80.2102	6445545	5019	polygon	FALSE	TRUE	America/Ne	1	33129 33125	1840015149
6	Dallas	Dallas	TX	Texas	48113	Dallas	32.7936	-96.7662	5743938	1526	polygon	FALSE	TRUE	America/Ch	1	75287 75098	1840019440
7	Philadelphia	Philadelphia	PA	Pennsylvani	42101	Philadelphia	40.0077	-75.1339	5649300	4554	polygon	FALSE	TRUE	America/Ne	1	19154 19151	1840000673
8	Houston	Houston	TX	Texas	48201	Harris	29.7863	-95.3889	5464251	1399	polygon	FALSE	TRUE	America/Ch	1	77069 77068	1840020925
9	Atlanta	Atlanta	GA	Georgia	13121	Fulton	33.7627	-84.4224	5449398	1441	polygon	FALSE	TRUE	America/Ne	1	30334 30331	1840013660
10	Washington	Washington	DC	District of C	11001	District of C	38.9047	-77.0163	5379184	4457	polygon	FALSE	TRUE	America/Ne	1	20010 20011	1840006060
11	Boston	Boston	MA	Massachuse	25025	Suffolk	42.3188	-71.0846	4688346	5532	polygon	FALSE	TRUE	America/Ne	1	02120 02121	1840000455
12	Phoenix	Phoenix	AZ	Arizona	4013	Maricopa	33.5722	-112.0891	4219697	1253	polygon	FALSE	TRUE	America/Ph	1	85008 85009	1840020568
13	Seattle	Seattle	WA	Washington	53033	King	47.6211	-122.3244	3789215	3469	polygon	FALSE	TRUE	America/Los	1	98109 98108	1840021117
14	San Francisc	San Francisc	CA	California	6075	San Francisc	37.7562	-122.443	3592294	7256	polygon	FALSE	TRUE	America/Los	1	94130 94131	1840021543

2. **Format** : CSV (>1000 records [≈ 28000])
3. **Data access and caching**: downloaded directly without additional verification methods. I used cache.
4. **Summary of data** : [28399 I will use all of them]

   The main usage of this data source is to provide an effective direct state-city relationship for the flask app. What's more, this data source have a very useful definition for city, city id, state and state id. I can use it as a mark for each city and state.

Important fields:

- "city" : name of the city.
- "city_ascii" : ascii version. This is more general, and I will use it as each city's name.
- "state_id" : abbreviation for state. This is more convenient than the *"state_name"* and can be used as a mark of the state.
- "state_name" : full state name.
- *"id"* : unique id for each city, which can be the primary key for each city in the database.

5. **Evidence of caching**

```
source_path = './src/location/uscities.csv'
file_name = 'us_cities'
file_type = 'locations'
us_cities = cache.sync_cache(file_name, file_type)
if us_cities == {} or len(us_cities) == 0:
    us_cities = csv_helper.csv_to_dict(source_path)
    cache.save_cache(file_name, file_type, us_cities)
```

Using API key to get base information and do analysis

1. **Origin** : Documentation Request : `GET https://api.yelp.com/v3/businesses/search`

2. **Format** : JSON (Each request can only get up to 50 results. you still can only get up to 1000 results using multiple queries and combinations of the "limit" and "offset" parameters)
3. **Data access and caching** : The Yelp Fusion API uses private key authentication to authenticate all endpoints. I used cache.
4. **Summary of data** : [I will use [50, 1000] for each category of restaurant]

   What I'm trying to get from this API is the data of different categories of restaurants. Each request can get up to 50 results. In order to make the flask app faster, I decided to let each type of category of the restaurant only gets 50 records at most to draw the average rating bar plot. There are approximately 200 types of restaurants available in yelp in the US. Then, each plot needs nearly 10000 records of restaurants(The restaurant may not be completely unique, because the restaurant may have more than one category)

   Important fields :

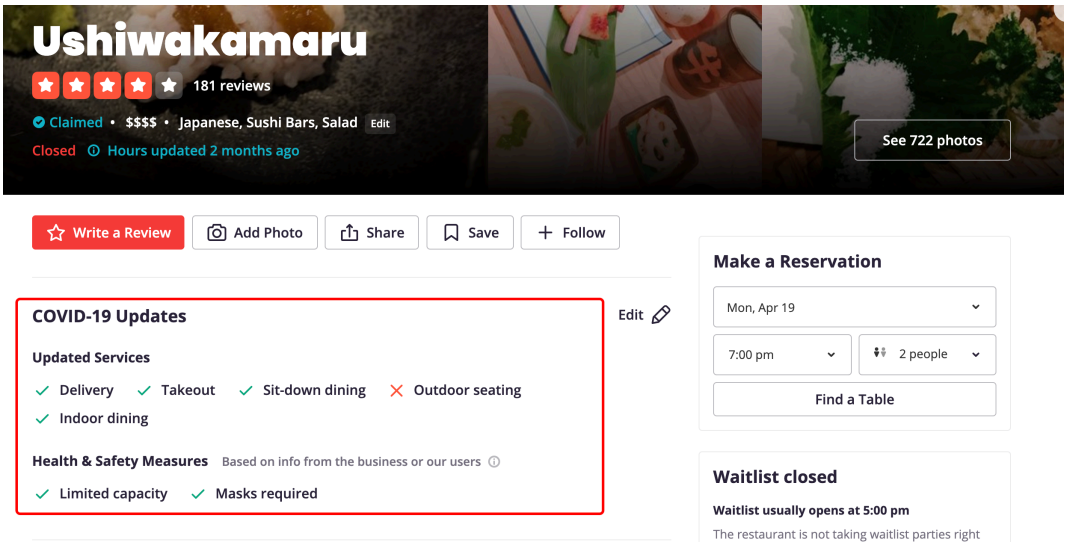
   - *"total"* : Total number of business Yelp finds based on the search criteria.
   - *"businesses"* : List of business Yelp finds based on the search criteria.
     - *"categories"* : List of category title and alias pairs associated with this business.
     - *"coordinates"* : Coordinates of this business.
     - *"id"* : Unique Yelp ID of this business. Example: `'4kMBvIEWPxWkWKFN__8SxQ'`
     - *"name"* : Name of this business.
     - *"rating"* : Rating for this business (value ranges from 1, 1.5, ... 4.5, 5).
5. **Evidence of caching**

```python
yelp_fusion_business_search_url = 'https://api.yelp.com/v3/businesses/search'
headers = {"Authorization": "Bearer {}".format(API_KEY)}

def api_search(category, location, offset=0):
    file_name = category + '_' + location + '_' + str(offset)
    file_type = 'restaurant_search'
    api_search = cache.sync_cache(file_name, file_type)
    if api_search == {}:
        params = {
            'categories' : category,
            'location' : location,
            'offset' : offset,
            'limit' : 50
            }
        r = requests.get(yelp_fusion_business_search_url, headers=headers, params=params)
        api_search = json.loads(r.text)
        cache.save_cache(file_name, file_type, api_search)
    return api_search
```

Crawling and scraping multiple pages in Yelp to gain information related covid-19

1. **Origin** : website : `https://www.yelp.com/`

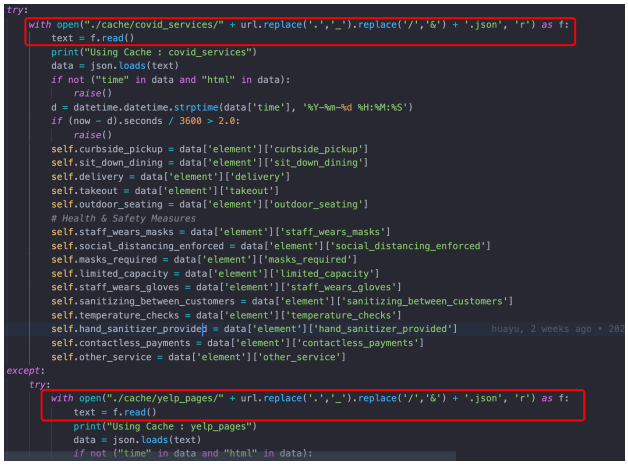


2. **Format** : Html
3. **Data access and caching** : By crawling and scraping. I used cache.
4. **Summary of data** : [I will use [50, 1000] for each category of restaurant]

   This part of the data is not fixed, each restaurant has its own services dealing with covid-19. There are some basic services provided by yelp. However, the user and also the owner of the restaurant can change the information on the website. I'm trying to get all of the services from the site and provide it in my flask app to the users.

   Important attributes :

   - *"Updated Services"* : some basic services the restaurant can provide to the customer.
   - *"Health & Safety Measures"* : what the health & satety measures the restaurant has implemented.
5. **Evidence of caching**



Database

Database schema

```
CREATE TABLE IF NOT EXISTS restaurant_category_information(
"title" TEXT NOT NULL, # name of the category
"alias" TEXT NOT NULL, # alias of the name
"country_whitelist" TEXT # Which countries offer searches in this category);
```

It contains all the restaurant categories (not all the categories provided by yelp fusion)

```
CREATE TABLE IF NOT EXISTS iso_3166_1_alpha_2_code(
"Code"  TEXT NOT NULL, # ISO 3166-1 alpha-2 code
"Name"  TEXT NOT NULL, # English short name officially used by the ISO 3166
        PRIMARY KEY("Code"));
```

ISO 3166-1 alpha-2 codes are two-letter country codes defined in ISO 3166-1, part of the ISO 3166 standard published by the International Organization for Standardization (ISO), to represent countries, dependent territories, and special areas of geographical interest. This form is suitable for filtering which categories are available in which countries (although my program is only used in the United States).

```
CREATE TABLE IF NOT EXISTS us_states(
"city"  TEXT, # name of the city
"city_ascii"  TEXT, # ascii of the name of the city
"state_id"  TEXT, # alpha2 of the state (`'NY'` for New York)
"state_name"  TEXT, # name of the state (New York)
"county_fips" TEXT,
"county_name" TEXT,
"lat" REAL,
"lng" REAL,
"population"  INTEGER,
"density" INTEGER,
"source"  TEXT,
```

```
"military"  INTEGER,
"incorporated"  INTEGER,
"timezone"  TEXT,
"ranking" INTEGER,
"zips"  TEXT,
"id"  TEXT, # unique id for each city
PRIMARY KEY("id"));
```

The purpose of this form is to help users select a certain city in a certain state, and then facilitate the flask application to continue to deal with subsequent statistical problems of different categories of restaurant ratings. You can see it from the pic in the *Interaction and Presentation Plans* part. Because location is needed when searching, all the location is associated with the search record table (restaurant_category_fetch).

```
CREATE TABLE IF NOT EXISTS restaurant_information(
"id"  TEXT, # id of each restaurant
"alias" TEXT, # alias of the restaurant
"name"  TEXT, # name of the restaurant
"image_url" TEXT, # url of the image of the restaurant
"is_closed" INTEGER, # whether is closed
"url" TEXT, # url in yelp
"review_count"  INTEGER,
"categories"  TEXT,
"rating"  REAL,
"coordinates_latitude"  REAL,
"coordinates_longitude" REAL,
"transactions"  TEXT,
"price" TEXT,
"location"  TEXT,
"phone" TEXT,
"display_phone" TEXT,
PRIMARY KEY("id"));
```

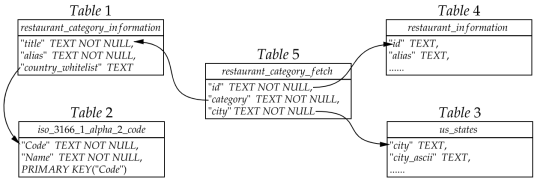
This table is used to record restaurant information. Through the search api provided by yelp fusion, information related to the search results can be collected. I will store the restaurant information obtained by each search in the database, because restaurant information is not frequently updated data.

```
CREATE TABLE IF NOT EXISTS restaurant_category_fetch (
"id"  TEXT NOT NULL, # id of restaurant(the same as restaurant_information.id)
"category"  TEXT NOT NULL, # searched category(the same as us_states.city_ascii)
"city" TEXT NOT NULL # searched city(the same as restaurant_category_information.title));
```

This is a table that records each search, which contains necessary information including location, category, and returned shop id. They are respectively associated with the three tables, please see the follow-up for details. This table may change in the future, or there is another table to filter the information of the categories of valid stores.

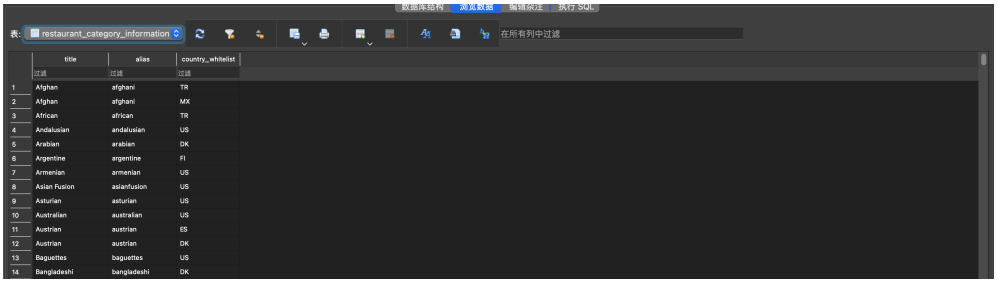
Foreign key-primary key relations

```
1. restaurant_category_information.country_whitelist = iso_3166_1_alpha_2_code.Code
2. restaurant_category_fetch.id = restaurant_information.id
3. restaurant_category_fetch.city = us_states.city_ascii
4. restaurant_category_fetch.category = restaurant_category_information.title
```

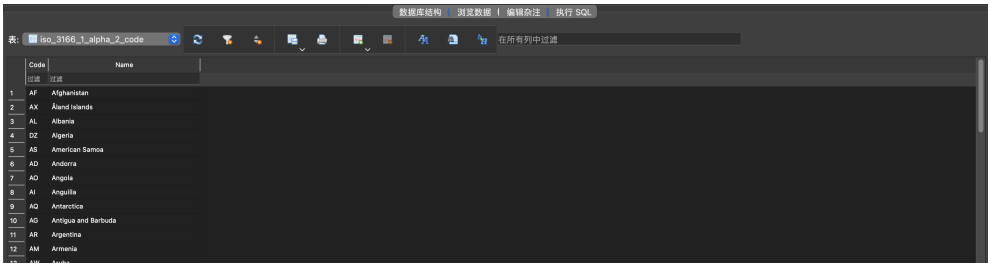


Screenshots of the data

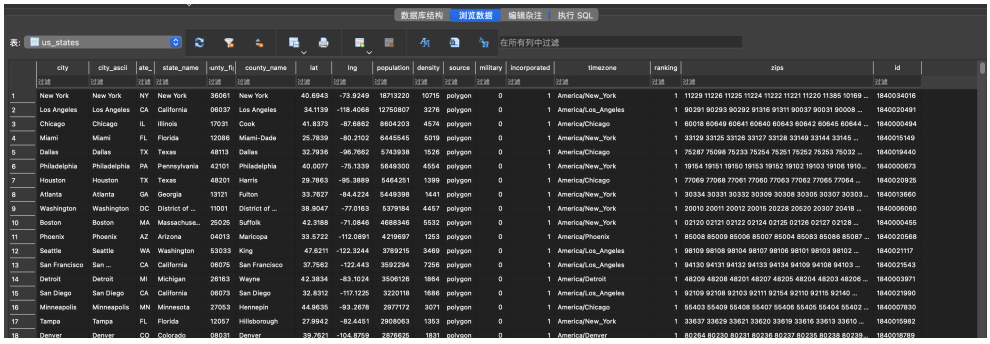
- Table 1 : restaurant_category_information



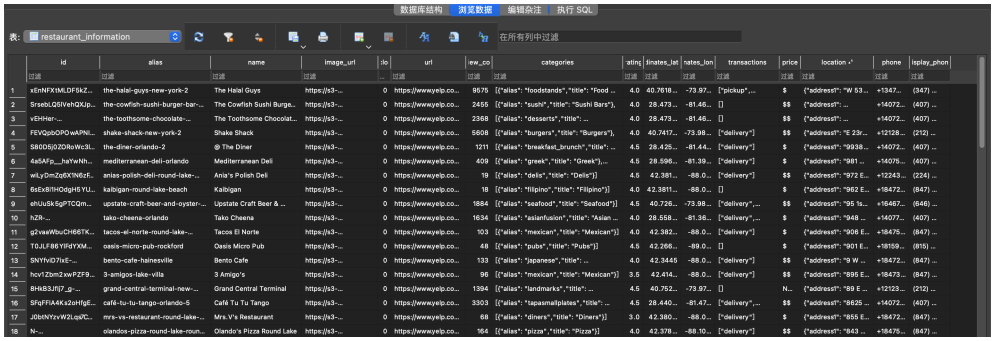
- Table 2 : iso_3166_1_alpha_2_code



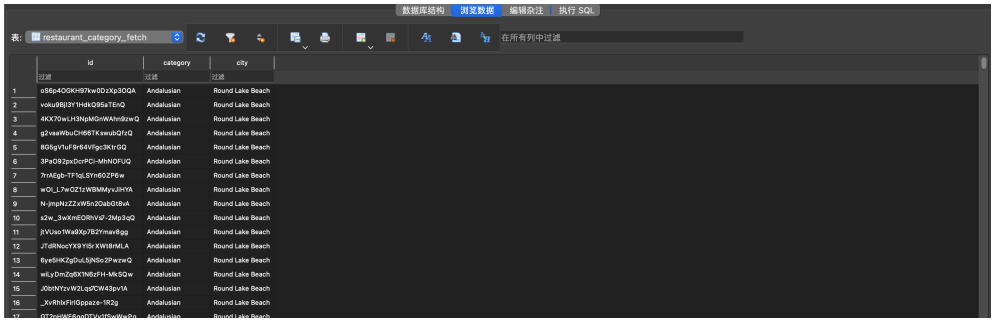
- Table 3 : us_states



- Table 4 : restaurant_information



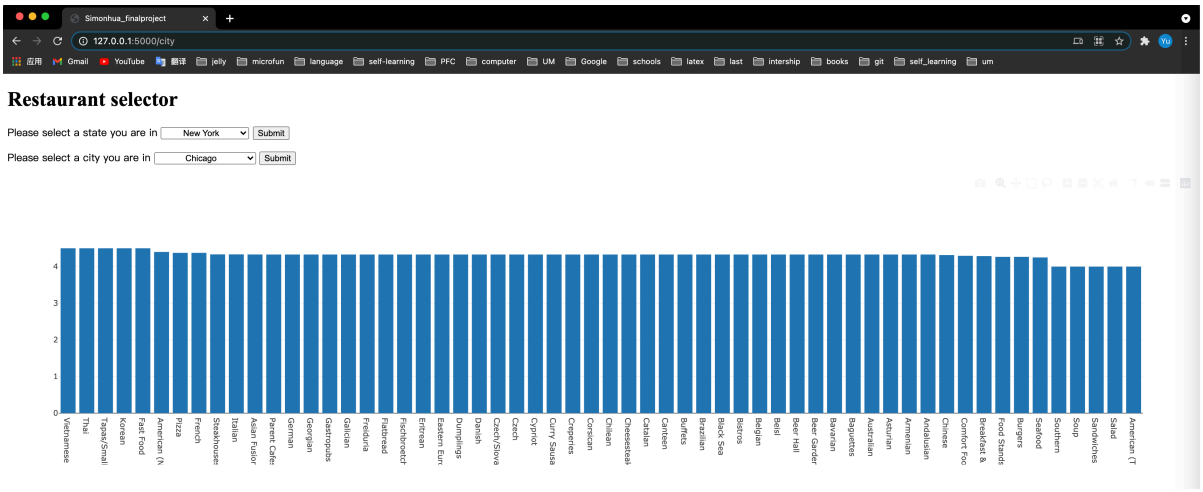
- Table 5 : restaurant_category_fetch



Interaction and Presentation Plans

Description

A program that lets a user choose a specific state and a city and see the average ratings for different restaurant types from Yelp as **plotly bar**. You can see it from the pic below. However, This is only a draft version and not the final version.You can select a state at first and submit, then a city and submit. The program will generate the bar plot by the state and the city you provided.



Then, you can choose a specific type of food, the program will present a useful statistical information about the covid-19 and specific restaurant information presented in a **table** to help you decide where to go. I will only show some useful information in the table.

The program will also provide a **plot of map** to facilitate you to choose a location closer to you. I intend to generate a map information from the address of each restaurant, so that users can roughly get the location of each restaurant. Specific information can be viewed by entering the detailed information of the restaurant.

Finally, you can choose one restaurant specifically, the program will offer you more details in **text**. This part will contain as much information as I can provide.

Technologies

1. **Flask** : The whole program is running within a Flask App.
2. **Plotly** : I used plotly to draw bar plot. There are some statistic data for covid 19 I can use.
3. **Command line** : Only for logging, you can see what the flask app is doing from the command line.