

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"  
    ]  
  },  
]
```

1. Format : JSON(> 1000 records)

2. Data access and caching : downloaded directly without additional verification methods, I used cache.

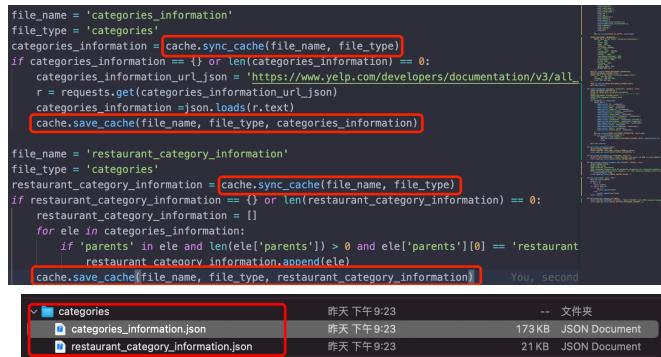
3. Summary of data : [≈ 1500 | I will use 192]

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)

4. 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_categories.json'  
    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": "AR", "Name": "Argentina"}, {"Code": "AM", "Name": "Armenia"}, {"Code": "AO", "Name": "Antarctica"}, {"Code": "AG", "Name": "Antigua and Barbuda"}, {"Code": "AR", "Name": "Argentina"}, {"Code": "AM", "Name": "Armenia"}, {"Code": "AO", "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": "BW", "Name": "Bosnia and Herzegovina"}, {"Code": "BV", "Name": "Botswana"}, {"Code": "BN", "Name": "Brunei Darussalam"}, {"Code": "BT", "Name": "Brazil"}, {"Code": "IO", "Name": "British Indian Ocean Territory"}, {"Code": "BN", "Name": "Brunei Darussalam"}, {"Code": "BG", "Name": "Bulgaria"}, {"Code": "IO", "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": "KI", "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": "CR", "Name": "Costa Rica"}, {"Code": "CI", "Name": "C\u00f4te d'Ivoire"}, {"Code": "HR", "Name": "Croatia"}, {"Code": "CU", "Name": "Cuba"}, {"Code": "CW", "Name": "Curacao"}, {"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": "Guernsey"}]
```

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

```

file_name = 'iso_3166_1_alpha_2_code'
file_type = 'locations'
iso_3166_1_alpha_2_code = cache.sync_cache(file_name, file_type)
if iso_3166_1_alpha_2_code == {} or len(iso_3166_1_alpha_2_code) == 0:
    iso_3166_1_alpha_2_code_url_json = "https://datahub.io/core/country-list/r/data.json"
    r = requests.get(iso_3166_1_alpha_2_code_url_json)
    iso_3166_1_alpha_2_code = json.loads(r.text)
    cache.save_cache(file_name, file_type, iso_3166_1_alpha_2_code)

locations
iso_3166_1_alpha_2_code.json
us_cities.json

```

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	cty_ascii	state_id	state_name	county	tips	county_nam	lat	lng	population	density	source	military	incorporate/timezone	ranking	zips	id
2	New York	New York	NY	New York	6007	20001	New York	40.6948	-74.9249	80713200	10715	polygon	FALSE	TRUE	America/Ne	111229	11226184000404010
3	Los Angeles	Los Angeles	CA	California	6007	20001	Los Angeles	34.0519	-118.4065	12700000	3276	polygon	FALSE	TRUE	America/Los	100900	9000184000040401
4	Chicago	Chicago	IL	Illinois	17031	Cook	Chicago	41.8373	-87.6662	8604203	4579	polygon	FALSE	TRUE	America/On	60918	692461840000404
5	Miami	Miami	FL	Florida	12096	Miami-Dade	Miami	25.7639	-80.2102	6445445	5019	polygon	FALSE	TRUE	America/Ne	33129	3312518400015149
6	Dallas	Dallas	TX	Texas	48113	Dallas	Dallas	32.7936	-96.7662	5749398	1526	polygon	FALSE	TRUE	America/Ch	75287	75096184001940
7	Philadelphia	Philadelphia	PA	Pennsylvania	42101	Philadelphia	Philadelphia	40.0077	-75.1339	5649300	4554	polygon	FALSE	TRUE	America/Ne	19154	191511840000673
8	Houston	Houston	TX	Texas	48201	Harris	Houston	29.7863	-95.3889	5464251	1399	polygon	FALSE	TRUE	America/Ch	77069	770661840020925
9	Atlanta	Atlanta	GA	Georgia	13121	Fulton	Atlanta	33.7627	-84.4224	5449398	1441	polygon	FALSE	TRUE	America/Ne	130334	303311840013660
10	Washington	Washington	DC	District of C	11001	District of C	Washington	38.9047	-77.0163	5379184	4467	polygon	FALSE	TRUE	America/Ne	20010	20011184000660
11	Boston	Boston	MA	Massachusetts	25025	Suffolk	Boston	42.3188	-71.0846	4688346	5532	polygon	FALSE	TRUE	America/Ne	102120	021211840000455
12	Phoenix	Phoenix	AZ	Arizona	4013	Maricopa	Phoenix	33.5722	-112.0891	4219697	1253	polygon	FALSE	TRUE	America/Ph	185008	850051840020568
13	Seattle	Seattle	WA	Washington	53033	King	Seattle	47.6211	-122.3244	3789215	3469	polygon	FALSE	TRUE	America/Los	198109	981081840021117
14	San Francisco	San Francisc	CA	California	6075	San Francisc	San Francisco	37.7562	-122.443	3592294	7256	polygon	FALSE	TRUE	America/Los	94130	941311840021543

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)

locations
iso_3166_1_alpha_2_code.json
us_cities.json

```

Using API key to get base information and do analysis

1. Origin : [Documentation](#) Request : GET <https://api.yelp.com/v3/businesses/search>

```

{
    "time": "2021-04-19 12:05:44",
    "cache": {
        "businesses": [
            {
                "id": "H4jJ7XB3CetIr1pg56Cz0",
                "alias": "levain-bakery-new-york",
                "name": "Levain Bakery",
                "image_url": "https://s3-media2.fl.yelpcdn.com/bphoto/jcdrXah-NjPa0Lb-30BWu0.jpg",
                "is_closed": false,
                "url": "https://www.yelp.com/biz/levain-bakery-new-york?adjust_creative=BE42nNx4TWWw@mcaxy9tW&utm_campaign=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=BE42nNx4TWWw@mcaxy9tW",
                "review_count": 8340,
                "categories": [
                    {
                        "alias": "bakeries",
                        "title": "Bakeries"
                    }
                ],
                "rating": 4.5,
                "coordinates": {
                    "latitude": 40.779961,
                    "longitude": -73.980299
                },
                "transactions": [],
                "price": "$$",
                "location": {
                    "address1": "167 W 4th St",
                    "address2": "",
                    "address3": "",
                    "city": "New York",
                    "zip_code": "10023"
                },
                "country": "US",
                "state": "NY",
                "display_address": [
                    "167 W 4th St",
                    "New York, NY 10023"
                ],
                "phone": "+19174643769",
                "display_phone": "(917) 464-3769",
                "distance": 8369.262424660568
            },
            {
                "id": "V7LXZKBDzScDeGB8JmzSA",
                "alias": "Katzs-delicatessen-new-york",
                "name": "Katz's Delicatessen",
                "image_url": "https://s3-media4.fl.yelpcdn.com/bphoto/gr7eSU6CFwRGZ7Rc-0EoTQ/"
            }
        ]
    }
}

```

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

```
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
```

文件名	修改时间	大小	类型
American (New)_Los Angeles_0.json	昨天下午9:26	51 KB	JSON Document
American (Traditional)_Los Angeles_0.json	昨天下午9:27	51 KB	JSON Document
Andalusian_Los Angeles_0.json	昨天下午9:24	51 KB	JSON Document
Armenian_Los Angeles_0.json	昨天下午9:24	51 KB	JSON Document
Asian Fusion_Los Angeles_0.json	昨天下午9:24	51 KB	JSON Document
Asturian_Los Angeles_0.json	昨天下午9:24	51 KB	JSON Document
Australian_Los Angeles_0.json	昨天下午9:24	51 KB	JSON Document

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

1. Origin : website : <https://www.yelp.com/>

COVID-19 Updates

Updated Services

- ✓ Delivery
- ✓ Takeout
- ✓ Sit-down dining
- ✗ Outdoor seating
- ✓ Indoor dining

Health & Safety Measures Based on info from the business or our users

- ✓ Limited capacity
- ✓ Masks required

Make a Reservation

Mon, Apr 19

7:00 pm

2 people

Find a Table

Waitlist closed

Waitlist usually opens at 5:00 pm

The restaurant is not taking waitlist parties right

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 & safety measures the restaurant has implemented.

5. Evidence of caching

```
try:
    with open("./cache/covid_services/" + url.replace('.', '_').replace('/', '&') + '.json', 'r') as f:
        text = f.read()
        print("Using Cache : covid_services")
        data = json.loads(text)
        if not ("time" in data and "html" in data):
```

```

        'other_service' : self.other_service
    )
data_covid_services_json = {"time": str(now)[0:19], "element": element}
with open("./cache/covid_services" + url.replace('/', '_').replace('&', '&') + '.json', 'w', encoding='utf-8') as f:
    f.write(json.dumps(data_covid_services_json, indent = 4, ensure_ascii = False))

def updateElement(self, name, status):
    name_lower = name.lower()
    if 'curbside pickup' in name_lower or 'curbside' in name_lower:

```

Database

Database schema

```

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

```

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

```

1 | CREATE TABLE IF NOT EXISTS iso_3166_1_alpha_2_code(
2 |     "Code" TEXT NOT NULL, # ISO 3166-1 alpha-2 code
3 |     "Name" TEXT NOT NULL, # English short name officially used by the ISO 3166
4 |         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).

```

1 | CREATE TABLE IF NOT EXISTS us_states(
2 |     "city" TEXT, # name of the city
3 |     "city_ascii" TEXT, # ascii of the name of the city
4 |     "state_id" TEXT, # alpha2 of the state (`'NY` for New York)
5 |     "state_name" TEXT, # name of the state (New York)
6 |     "county_fips" TEXT,
7 |     "county_name" TEXT,
8 |     "lat" REAL,
9 |     "lng" REAL,
10 |    "population" INTEGER,
11 |    "density" INTEGER,
12 |    "source" TEXT,
13 |    "military" INTEGER,
14 |    "incorporated" INTEGER,
15 |    "timezone" TEXT,
16 |    "ranking" INTEGER,
17 |    "zips" TEXT,
18 |    "id" TEXT, # unique id for each city
19 |    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).

```

1 | CREATE TABLE IF NOT EXISTS restaurant_information(
2 |     "id" TEXT, # id of each restaurant
3 |     "alias" TEXT, # alias of the restaurant
4 |     "name" TEXT, # name of the restaurant
5 |     "image_url" TEXT, # url of the image of the restaurant
6 |     "is_closed" INTEGER, # whether is closed
7 |     "url" TEXT, # url in yelp
8 |     "review_count" INTEGER,
9 |     "categories" TEXT,
10 |    "rating" REAL,
11 |    "coordinates_latitude" REAL,
12 |    "coordinates_longitude" REAL,
13 |    "transactions" TEXT,
14 |    "price" TEXT,
15 |    "location" TEXT,
16 |    "phone" TEXT,
17 |    "display_phone" TEXT,
18 |    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.

```

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

```

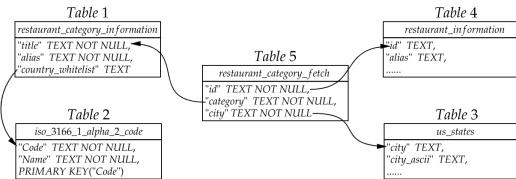
This is a table that records each search, which contains necessary information including location, category, and returned shop id. They are respectively associated with the three tables, please see the follow-up for details. This table may change in the future, or there is another table to filter the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

Foreign key-primary key relations

```

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

```



Screenshots of the data

- Table 1 : restaurant_category_information

	title	alias	country_whitelist
1	Afghan	afghani	TR
2	Afghani	afghani	MX
3	Afghanistan	afghan	TR
4	Anatolian	anatolian	US
5	Arabian	arabian	DK
6	Argentine	argentine	PT
7	Armenian	armenian	US
8	Asian Fusion	asianfusion	US
9	Auturian	auturian	US
10	Australian	australian	US
11	Austrian	austrian	ES
12	Austrian	austrian	DK
13	Baguettes	baguettes	US
14	Bangladeshi	bangladeshi	DK

- Table 2 : iso_3166_1_alpha_2_code

	Code	Name
1	AF	Afghanistan
2	AX	Aland Islands
3	AL	Albania
4	DZ	Algeria
5	AS	American Samoa
6	AD	Andorra
7	AO	Angola
8	AI	Anguilla
9	AQ	Antarctica
10	AG	Antigua and Barbuda
11	AR	Argentina
12	AM	Armenia
13	AW	Aruba

- Table 3 : us_states

	id	city	city_ascii	state	state_name	unity_fi	county_name	lat	long	population	density	source	incorporated	timezone	ranking	zips	
1	New York	New York	New York	36061	New York	40.6943	-73.9249	16173220	1075 polygon	0	1	America/New_York	11229 11228 11225 11224 11222 11221 11220 11383 10169...	1840034016			
2	Los Angeles	Los Angeles	CA California	60637	Los Angeles	34.1159	-118.4068	1759807	3276 polygon	0	1	America/Los_Angeles	90293 90292 90291 91316 91316 90037 90031 90000...	1840020491			
3	Chicago	Chicago	IL Illinois	70701	Cook	41.8737	-87.6662	8604203	4574 polygon	0	1	America/Chicago	8009 80049 80048 80040 80040 80043 80042 80045 80044...	1840000484			
4	Miami	Miami	FL Florida	12086	Miami-Dade	26.7839	-80.2102	6445545	5019 polygon	0	1	America/New_York	3132 31325 31324 31327 31323 31349 31314 31315...	1840015149			
5	Dallas	TX Texas	48110	Dallas	32.7935	-96.7665	5743938	1526 polygon	0	1	America/Chicago	7287 76098 72323 72324 72325 72325 72323 79302...	1840019440				
6	Philadelphia	Philadelphia	PA Pennsylvania	42101	Philadelphia	40.0077	-75.1339	5648300	4554 polygon	0	1	America/New_York	19154 19151 19150 19153 19152 19102 19103 19102 1910...	1840000673			
7	Houston	Houston	TX Texas	48201	Harris	29.7863	-95.3889	5464251	1399 polygon	0	1	America/Chicago	17068 77098 77081 77077 77062 77065 77064...	1840020925			
8	Atlanta	Atlanta	GA Georgia	19321	Fulton	33.7627	-84.4224	5449398	1441 polygon	0	1	America/Chicago	30338 30331 30332 30330 30339 30335 30337 3033...	1840013680			
9	Washington	Washington	DC District of Columbia	13001	District of Columbia	38.9047	-77.0163	5379384	4487 polygon	0	1	America/New_York	1 2010 20011 20011 20011 20011 20229 20250 20307 20419...	1840006060			
10	Boston	Boston	MA Massachusetts	20225	Suffolk	42.3188	-71.0843	4688346	5537 polygon	0	1	America/New_York	1 2120 21211 21222 21212 21214 21212 21218 21217 21218...	1840000455			
11	Phoenix	Phoenix	AZ Arizona	40413	Maricopa	33.5722	-112.0891	4219897	1283 polygon	0	1	America/Phoenix	1 85008 85009 85000 85007 85004 85003 85008 85007...	1840020968			
12	Seattle	Seattle	WA Washington	53033	King	47.6211	-122.3244	3789215	3449 polygon	0	1	America/Los_Angeles	88108 88108 88107 88107 88101 88103 88102...	1840021117			
13	San Francisco	San Francisco	CA California	60675	San Francisco	37.7662	-122.444	3592294	7295 polygon	0	1	America/Los_Angeles	1 8410 94131 94132 94133 94134 94109 94108 94103...	1840021543			
14	Detroit	Detroit	MI Michigan	28163	Wayne	42.3834	-83.1024	3506126	1864 polygon	0	1	America/Detroit	1 4820 48208 48201 48207 48205 48203 48203 48202...	1840003971			
15	San Diego	San Diego	CA California	60673	San Diego	32.8312	-117.1228	3220198	1866 polygon	0	1	America/Los_Angeles	1 2120 92108 92103 92111 92105 92110 92115 92140...	1840021990			
16	Minneapolis	Minneapolis	MN Minnesota	27053	Hennepin	44.9635	-93.2679	2977172	3071 polygon	0	1	America/Chicago	1 85402 85409 85404 85407 85405 85408 85404 85402...	1840007830			
17	Tempe	Tempe	AZ Arizona	12057	Hillsborough	27.9942	-102.4481	2908063	1393 polygon	0	1	America/New_York	1 33637 33629 33626 33620 33619 33616 33613 33610...	1840015982			
18	Denver	Denver	CO Colorado	60831	Denver	39.7621	-104.8750	2876626	1831 polygon	0	1	America/Denver	1 80264 80230 80231 80238 80237 80235 80238 80239...	1840018789			

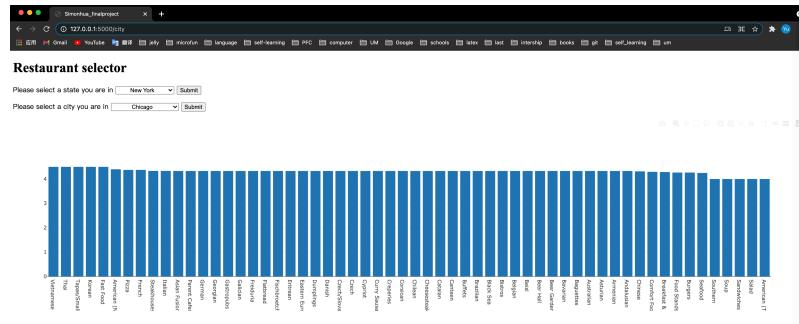
- Table 4 : restaurant_information

- 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.