P3: Wrangle OpenStreetMap Data

Map Area: Taipei, Taiwan
OpenStreetMap: Taipei City, Taiwan
MapZen: search for "Taipei, City, Taiwan"

Acknowledge
Taiwan postal code search API

It has been about 5 years since I left Taipei city where I had been living for 6 years. Therefore, I am interested in having an insight into the map data of Taipei, Taiwan. Actually, it contains some area of New Taipei City (old name Taipei County), which is reasonable that Taipei City and New Taipei City are always discussed together. And Taipei-Keelung metropolitan area includes Taipei City, New Taipei City and Keelung City.

Section 1. Problems Encountered in the Map

The follows are the main problems I found. I would discuss each probelm and show my way to solve it in the following paragraphes.
- *Inconsistent address structure*
- *Inconsistent phone number*
- *Inconsistent postal code*
- *Inconsistent second-level k tag's value*
- *Second-level k tag's value is "type"*

In order to make the investigation more efficient, I would like to shape the address into following structure:

```
"address": { "city": "city name",
             "district": "district name",
             "village": "village name",
             "neighborhood": "neighborhood name",
             road": "road name",
             "street": "street name",
             "lane": "lane value",
             "alley": "alley value",
             "housenumber": "housenumber value",
             "floor": "floor value",
             "housename": "house name",
             "postcode": "postcode value"}
```

ex. "臺北市大安區羅斯福路四段 1 號" would be transformed as following:

```
"address": {"city": "臺北市",
            "district": "大安區",
            "road": "羅斯福路",
            "section": "4",
            "housenumber": "1",
            "postcode": "60625"}
```

- Inconsistent address structure
- *Address tag in breakdown structure only.*
- *Full address tag only.*
- *Some address tags lost in breakdown structure.*
- *City tag content the information of district tag.*
- *Street tag content the information of section tag.*

The difficulty of this problem is to separate full, city or district addresses into the appropriate terms.
ex. `<tag k="addr:full" v="10666 臺北市大安區羅斯福路四段 1 號"/>`
`(<tag k="addr:full" v="No. 1, Sec. 4th, Roosevelt Rd., Da'an Dist., Taipei City, 10666"/>)`

Besides, there is no `k="addr:road"` in the osm file. Thus, it is necessary to check if it is road or street. For the case of that some address tags lost in the breakdown structure, the full address would be broken down to insert into the dictionary if it exists. Furthermore, some city tags and road/street tags contain the other information not belongs to them.

ex. `<tag k="addr:city" v="臺北市大安區"/>` or `<tag k="addr:street" v="羅斯福路四段"/>`
(`<tag k="addr:city" v="Da'an Dist. Taipei City"/>`,
`<tag k="addr:street" v="Sec. 4th, Roosevelt Rd."/>`)

The former should be separated into "臺北市" and "大安區", the latter should be separated into "羅斯福路" and "四段". In these cases, it is necessary to separate them into the appropriate terms. Actually, I also fund some road/street tags content the information of lane, alley and so on. The results indicate that it is necessary to write a function to solve this problem because the amount is not few.

```
> db.mycol.aggregate([{'$match': {'address.city': {'$exists': 1}}},
                      {'$group': {'_id': '$address.city', 'count': {'$sum':1}}},
                      {'$sort': {'count': -1}}])

{ "_id" : "臺北市", "count" : 1077 }
{ "_id" : "台北市", "count" : 951 }
{ "_id" : "新北市", "count" : 337 }
{ "_id" : "新北市板橋區", "count" : 109 }
{ "_id" : "台北市內湖區", "count" : 43 }
{ "_id" : "臺北市北投區", "count" : 35 }
{ "_id" : "Taipei", "count" : 35 }
{ "_id" : "台北市中山區", "count" : 29 }
{ "_id" : "新北市永和區", "count" : 23 }
{ "_id" : "台北市士林區", "count" : 23 }
{ "_id" : "新北市新店區", "count" : 20 }
{ "_id" : "台北市大安區", "count" : 17 }
{ "_id" : "新北市中和區", "count" : 16 }
{ "_id" : "台北市中正區", "count" : 16 }
{ "_id" : "台北市大同區", "count" : 14 }
{ "_id" : "台北市松山區", "count" : 13 }
{ "_id" : "臺北市士林區", "count" : 11 }
{ "_id" : "台北市文山區", "count" : 11 }
{ "_id" : "台北市南港區", "count" : 10 }
{ "_id" : "新北市五股區", "count" : 9 }
{ "_id" : "臺北", "count" : 8 }
{ "_id" : "Taipei City", "count" : 8 }
{ "_id" : "台北市北投區", "count" : 6 }
{ "_id" : "新北市三重區", "count" : 6 }
{ "_id" : "松山區", "count" : 5 }
{ "_id" : "板橋區", "count" : 5 }
{ "_id" : "台北", "count" : 5 }
{ "_id" : "北投區", "count" : 4 }
{ "_id" : "土城區", "count" : 4 }
{ "_id" : "臺北市大同區", "count" : 3 }
{ "_id" : "台北市(Taipei City)" , "count" : 3 }
{ "_id" : "新北市蘆洲區", "count" : 3 }
{ "_id" : "臺北市大安區", "count" : 3 }
Type "it" for more
>
```

Eventually, I collected the information of postal code and `"address:full"` only if it exists. Otherwise, all the address tag would be collected. And then I broke down all the address information to create address dictionary like the structure shown above.

Meanwhile, valid postal code was collected from `"address:postcode"` first or `"address:full"`. If both information does not exist, then the postal code will be obtained by API, which will be discussed below.

- Inconsistent phone number

(+886: country code for Taiwan, (0)2: area code for Taipei)

- The formats of the phone number are inconsistent.
  ex. +886-2-2345-6789, +886 2 2345 6789, +886 2 2345-6789 and so on.
- Some phone numbers are the mobile phone number.
  ex. +886 937672988
- Some nodes have more than one phone number.
  ex. `<tag k="phone" v="+886-2-1234-5678; +886 2 2345 6789"/>`

In order to fix this problem, I arranged the phone numbers into an array and reformatted them as follows:

ex. `{"phone": ["+886-2-1234-5678", "+886-2-2345-6789"]}`

To confirm the function can treat all the phone numbers well, I confirmed the data by following query:

```
> db.mycol.aggregate([{'$match': {'phone':{'$exists':1}}},
                      {'$group': {'_id':'$phone','count':{'$sum':1}}},
                      {'$sort':{'count':-1}}])
{ "_id" : "+886975551156", "count" : 7 }
{ "_id" : "+886-2-2381-3135", "count" : 4 }
{ "_id" : "+886(2)27126390", "count" : 4 }
{ "_id" : "+886(2)25702330#6199", "count" : 3 }
{ "_id" : "+886-2-27274168", "count" : 2 }
{ "_id" : "+886 2 2348 3456", "count" : 2 }
{ "_id" : "+886(2)27325315", "count" : 2 }
{ "_id" : "+886 2 89696550", "count" : 2 }
{ "_id" : "+886-2-2322-8000", "count" : 2 }
{ "_id" : "+886(9)22323366", "count" : 1 }
{ "_id" : "+886(02)88664433", "count" : 1 }
{ "_id" : "+886(2)28963910", "count" : 1 }
{ "_id" : "+886-2-66250899", "count" : 1 }
{ "_id" : "+886-2-2579-2065", "count" : 1 }
{ "_id" : "+886 2 2523 5000", "count" : 1 }
{ "_id" : "+886-2-27135211", "count" : 1 }
{ "_id" : "+886255713333", "count" : 1 }
{ "_id" : "+886-2-25519321", "count" : 1 }
{ "_id" : "+886-2-2397-1377", "count" : 1 }
Type "it" for more
>
```

Besides, I also found some phone numbers with extension code as follows, and their formats are not consistent. This problem would be solved in audit process.

```
{ "_id" : "+886(2)25702330#6199", "count" : 3 }
{ "_id" : "+886 2 1234 5678;02 12345678;+886-937123456;+886(02)12345678", "count" : 1 }
{ "_id" : "(02)2236-8225#63302、63304", "count" : 1 }
{ "_id" : "+886-2-33169908 轉 0252", "count" : 1 }
```

轉 means extend.

- Inconsistent postal code

The postal code has 5 digits call 3+2 postal code in Taiwan. The former 3 digits indicate the administrative district,
and the latter 2 digits indicates special use or the specific area in a district.
Basically, the former 3 digits is enough for the delivery, so that 3-digit postal codes still remain.
To solve this problem, I found an API which return the 3+2 postal code with providing the appropriate address.
The appropriate address has to content city, district, street and house number (as below example).
Fortunately, it returns the information in JSON format, which is convenient for me.

request:
```
http://zip5.5432.tw/zip5json.py?adrs=臺北市大安區羅斯福路四段 1 號
```

respond:
```
{"dataver": "10312",
"adrs": "10617 臺北市大安區羅斯福路四段 1 號",
"zipcode": "10617",
"new_adrs2": "10617 臺北市大安區羅斯福路四段 1 號",
"new_adrs": "10617 臺北市大安區羅斯福路四段 1 號"}
```

However, one should be notified is that the osm file contents the information of the administrative district. It means that it is correct for them with 3-digit postal code. Fortunately, the documents of the administrative district do not have the "address" dictionary. Thus, postcode update would be carried out for "address" dictionary only.

- Second-level tag k's value is "type"

My python codes used to wrangle the data were based on those of problem set 6.
Therefore, second-level tag with that k's value is **"type"** would override the **node["type"]**.
I found this problem when I double check the data I inserted into MongoDB with following query.

```
> db.mycol.find({'type': 'node'}).count()
307488
> db.mycol.find({'type': 'way'}).count()
52325
```

However, the results are different from that of mapparser.py whoes results as follows:

```
{'bounds': 1,
 'member': 55822,
 'nd': 367968,
 'node': 307499,
 'osm': 1,
 'relation': 3248,
 'tag': 250687,
 'way': 52331}
```

Then, count the documents whoes type is either 'node' nor 'way' by following query.

```
> db.mycol.find({'type': {'$nin':['node', 'way']}}).count()
17
```

To solve this problem, the simple way is to change the key value if the tag k's value = **"type"**. In order to decide new key value, I checked tag k's value of these 23 documents by following query.

```
> db.mycol.aggregate([{'$match': {'type': {'$nin' :['node', 'way']}}},
                      {'$group': {'_id': '$type','count': {'$sum': 1}}},
                      {'$sort': {'count': -1}}])
{ "_id" : "multipolygon", "count" : 2 }
{ "_id" : "water", "count" : 2 }
{ "_id" : "Chinese", "count" : 2 }
{ "_id" : "religion", "count" : 1 }
{ "_id" : "inner", "count" : 1 }
{ "_id" : "audio", "count" : 1 }
{ "_id" : "prep-school", "count" : 1 }
{ "_id" : "status", "count" : 1 }
{ "_id" : "freshwater;saltwater", "count" : 1 }
{ "_id" : "school", "count" : 1 }
{ "_id" : "tree_start_date", "count" : 1 }
{ "_id" : "ngo", "count" : 1 }
{ "_id" : "broad_leaved", "count" : 1 }
{ "_id" : "tree", "count" : 1 }
```

As the results shown above, it is hard to define a general value of the key. Eventually the decision is to change the key value to **"other_type"**.

Section 2. Overview of the Data

- File Size

```
taipei_city_taiwan.osm:  72M
taipei_city_taiwan.osm.json:  99M
```

- Basic Statistical Observation

- Number of documents
```
> db.mycol.find().count()
359830
```

- Number of nodes
```
> db.mycol.find({'type':'node'}).count()
307499
```

- Number of ways
```
> db.mycol.find({'type':'way'}).count()
52331
```

- Number of unique users
```
> db.mycol.distinct('created.user').length
1030
```

- Top 5 contributing users
```
> db.mycol.aggregate([{'$group':{'_id':'$created.user', 'count':{'$sum':1}}},
                      {'$sort':{'count':-1}}, {'$limit':5}])
{ "_id" : "Supaplex", "count" : 100845 }
{ "_id" : "siaoyo", "count" : 35854 }
{ "_id" : "Littlebtc", "count" : 25169 }
{ "_id" : "dongpo", "count" : 12842 }
{ "_id" : "Cicerone", "count" : 9088 }
```
- Number of users having 1 post only
```
> db.mycol.aggregate([{'$group':{'_id':'$created.user', 'count':{'$sum':1}}},
                      {'$group':{'_id':'$count', 'num_user':{'$sum':1}}},
                      {'$sort': {'_id':1}}, {'$limit':1}])
{ "_id" : 1, "num_user" : 184 }
```
*"_id": only the count of the posts.

Section 3. Other ideas about the datasets

- Idea of Address

In this project, I spend lots of time on processing the address, because there are many pattern of addresses. The following report would reveal the difference in the results between **`data.py`** (*original*) written in the lesson and modified **`data.py`** (*modified*) written for process the information of the addresses.

- Number of Address
  - *original*
    ```
    > db.original.find({'address':{'$exists':1}}).count()
    5816
    ```
  - *modified*
    ```
    > db.modified.find({'address':{'$exists':1}}).count()
    5814
    ```
- Number of postal code
  - *original*
    ```
    > db.original.find({'address.postcode':{'$exists':1}}).count()
    1982
    ```
  - *modified*
    ```
    > db.modified.find({'address.postcode':{'$exists':1}}).count()
    1032
    ```

- Number of valid postal code
  - *original*
    ```
    > db.original.find({'address.postcode':{'$exists':1},
    ... '$where':'this.address.postcode.length == 5'}).count()
    1021
    ```
  - *modified*
    ```
    > db.modified.find({'address.postcode':{'$exists':1},
    ... '$where':'this.address.postcode.length == 5'}).count()
    1031
    ```
- Number of invalid postal code
  - *original*
    ```
    > db.original.find({'address.postcode':{'$exists':1},
    ... '$where':'this.address.postcode.length != 5'}).count()
    961
    ```
  - *modified*
    ```
    > db.modified.find({'address.postcode':{'$exists':1},
    ... '$where':'this.address.postcode.length != 5'}).count()
    1
    ```
- List of unique city
  - *original*
    ```
    > db.original.distinct('address.city')
    ["臺北市","新北市三重區","台北市","Taipei","新北市","新北市板橋區","土城區","台北市南港區
    ","台北市內湖區","台北市士林區","Taipei City","台北市中山區","Taipai","台北市大安區","
    臺北市大安區","新北市 三重區, Sanchong Dist., New Taipei City","台北市萬華區","松山區
    ","台北市北投區","新北市永和區","台北","新北市新店區","台北市松山區","新北市中和區","台北
    市中正區","新北市汐止區","板橋區","台北市文山區","台北市信義區","台北市大同區","臺北市信義
    區","板橋","新北市五股區","台北市(Taipei City)" ,"台北市 文山區","臺北市中山區","61 號
    ","臺北市萬華區","臺北","新北市板橋區四川路 1 段 287 號","新北市新北市","新北•市板橋區","
    新北市新莊區","新北市蘆洲區","士林區","北投區","三重區","臺北市北投區","臺北市士林區","中
    和市","臺北市松山區","新北是","台北市大同區迪化街一段 155 號","台北市大同區迪化街一段 156 號
    ","台北市大同區迪化街一段 147 號","台北市大同區迪化街一段 143 號","臺北市南港區","臺北市大同
    區","臺北市文山區","中和區","中正區","台北市__","新店區, 新北市","新北市內湖區","新北市土
    城區"]
    ```
  - *modified*
    ```
    > db.modified.distinct('address.city')
    ["臺北市", "新北市", null, "中和市"]
    ```
- List of unique administrative district
  - *original*
    ```
    > db.original.distinct('address.district')
    ["中正區", "北投區", "士林區", "文山區", "大同區", "內湖區", "大安區", "萬華區", "南港區
    ", "中山區", "板橋區", "新店區", "信義區", "南港", "中和區", "蘆洲區", "深坑區", "五股區
    ", "松山區", "淡水區", "大安", "三重區", "土城區"]
    ```
  - *modified*
    ```
    > db.modified.distinct('address.district')
    ["中正區", "北投區", "士林區", "文山區", "三重區", "大同區", "內湖區", "大安區", "萬華區
    ", "南港區", "中山區", "板橋區", "汐止區", "土城區", "新店區", "信義區", "中和區", "松
    山區", "永和區", "新莊區", "蘆洲區", "五股區", "淡水區", "深坑區"]
    ```

According to the results shown above, only two addresses could not pass my address process, which have not enough information to create the dictionary.
About the results of postal code, there are 961 invalid postal code, the digit number not equals to 5, in the original data. If we look at the results of valid postal code, there are 10 postal code which was extracted from the full address. On the other hand, the results of invalid postal code show that there is 1 result of modified **`data.py`**. I picked up that document from MongoDB as follow:

```
> db.modified.find({'address.postcode':{'$exists':1},
... '$where':'this.address.postcode.length != 5'})
{ "_id" : ObjectId("5699ec83a7d33d1092cc1924"), "website" : "http://www.joseph8022.com.tw/",
"amenity" : "kindergarten", "name" : "若石幼兒園", "created" : { "changeset" : "32736063",
"user" : "Supaplex", "version" : "3", "uid" : "274857", "timestamp" : "2015-07-19T16:26:07Z" },
```

"pos" : [25.0071837, 121.53908], "phone" : ["+886-2-2932-8022"], "address" : { "postcode" : "3" }, "type" : "node", "id" : "3621381691" }

It is obvious that there is something wrong with processed address which is `"address" : { "postcode" : "3" }`. Thus, it is necessary to investigate the original OSM file. The original data is follow:

```
<node id="3621381691" lat="25.0071837" lon="121.53908" version="3" timestamp="2015-07-
19T16:26:07Z" changeset="32736063" uid="274857" user="Supaplex">
    <tag k="name" v="若石幼兒園"/>
    <tag k="phone" v="+886-2-2932-8022"/>
    <tag k="amenity" v="kindergarten"/>
    <tag k="website" v="http://www.joseph8022.com.tw/"/>
    <tag k="addr:city" v="台北市"/>
    <tag k="addr:full" v="3"/>
    <tag k="addr:street" v="羅斯福路五段"/>
    <tag k="addr:housenumber" v="51"/>
  </node>
```

Corresponding v's value of `addr:full` is `"3"`. However, my processing algorithm only consider `addr:full` if it exists, and `if isInt(address['full'][:5]):` is used to check whether the first 5 characters is numeric or not. So, these reasons result in this problem. Fortunately, there is only one document has this problem which I can fix it by hand. So, the correct processed address should be as follow:

```
"address": { "city": "臺北市",
             "road": "羅斯福路",
             "section": "五",
             "housenumber": "51"}
```

Besides, I have to revise my python code to check whether the information of `addr:full` is valid or not. If not, it is necessary to collect the information of other address if they exist. And double check the length of first 5 characters extracted from `addr:full` even it is numeric.
Actually, the results of modified data.py did not update the postal code by API. The results of data.py with API is as follows:

- Number of postal code
  - *api*

```
> db.api.find({'address.postcode':{'$exists':1}}).count()
2163
```

- Number of valid postal code
  - *api*

```
> db.api.find({'address.postcode':{'$exists':1},
... '$where':'this.address.postcode.length == 5'}).count()
2163
```

- Number of valid postal code
  - *api*

```
> db.api.find({'address.postcode':{'$exists':1},
... '$where':'this.address.postcode.length != 5'}).count()
0
```

The above shows that the postal code obtain from the API is perfect. However, compared the results with the total number of address if exists, there are still over half of all the addresses with the problem like not enough information, so that the postal code can not be obtain by the API, which means over half of the addresses are not valid for the delivery.

Then, we continuously look at the results of city. Basically, it is no problem with the results of modified data.py. However, `"中和市"` should not appear because it is the old name of , and it is the administrative district of New Taipei City. The same, I picked up that document from MongoDB and investigated the data in the original OSM file as follows:

```
> db.modified.find({'address.city': "中和市"})
{ "_id" : ObjectId("5699ec84a7d33d1092cc523e"), "name" : "佑林醫院", "office" : "physician",
"created" : { "changeset" : "34115565", "user" : "Dolphinsue", "version" : "1", "uid" :
"3248756", "timestamp" : "2015-09-19T06:31:42Z" }, "pos" : [ 25.0057182, 121.4761215 ],
"phone" : [ "+886-222-239-293" ], "address" : { "city" : "中和市", "section" : "三",
"housenumber" : "71", "road" : "中山路" }, "type" : "node", "id" : "3750112414" }
```

```
<node id="3750112414" lat="25.0057182" lon="121.4761215" version="1" timestamp="2015-09-
19T06:31:42Z" changeset="34115565" uid="3248756" user="Dolphinsue">
    <tag k="name" v="佑林醫院"/>
    <tag k="phone" v="0222239293"/>
    <tag k="office" v="physician"/>
    <tag k="addr:city" v="中和市"/>
    <tag k="addr:street" v="中山路三段"/>
    <tag k="addr:postcode" v="235"/>
    <tag k="addr:housenumber" v="71"/>
  </node>
```

So, the reason is simple that the consideration of address process was not enough. I should think of that there should be old name of the administrative district in `addr:city` when I found that some `addr:city` contains the information of the administrative district.

- Additional data exploration using MongoDB queries

Besides the deep investigation of address, here show some additional idea about what kind of information can be explored by using MongoDB queries. The data by modified `data.py` with the API is used here.

- Top 5 user contributing to the OpenStreetMap

```
Pipeline:
    [{'$match': {'created.user':{'$exists':1}}},
     {'$group': {'_id':{'user':'$created.user'},'count':{'$sum':1}}},
     {'$project':{'_id':0, 'User':'$_id.user', 'count':'$count'}},
     {'$sort':{'count':-1}},
     {'$limit':5}]
Result:
[{User: Supaplex, count: 100845},
 {User: siaoyo, count: 35854},
 {User: Littlebtc, count: 25169},
 {User: dongpo, count: 12842},
 {User: Cicerone, count: 9088}]
```

- Top 5 user contributing to valid address (`address.postcode` exists)

```
Pipeline:
    [{'$match': {'created.user':{'$exists':1}}},
     {'$group': {'_id':{'user':'$created.user'},'count':{'$sum':1}}},
     {'$project':{'_id':0, 'User':'$_id.user', 'count':'$count'}},
     {'$sort':{'count':-1}},
     {'$limit':5}]
Result:
[{User: Supaplex, count: 647},
 {User: bananatw, count: 434},
 {User: twister88564, count: 367},
 {User: jiazheng, count: 87},
 {User: Vintagejhan, count: 60}]
```

- Top 5 amenities

```
Pipeline:
    [{'$match': {'amenity':{'$exists':1}}},
     {'$group': {'_id':{'amenity':'$amenity'},'count':{'$sum':1}}},
     {'$project':{'_id':0, 'amentiy':'$_id.amenity', 'count':'$count'}},
     {'$sort':{'count':-1}},
     {'$limit':5}]
Result:
[{User: restaurant, count: 2751},
 {User: parking, count: 824},
```

```
 {User: cafe, count: 618},
 {User: place_of_worship, count: 548},
 {User: bank, count: 487}]
```

- Top 5 restaurant cuisine

```
Pipeline:
    [{'$match': {'amenity':{'$exists':1}, 'cuisine': {'$exists':1}}},
     {'$group': {'_id': {'cuisine': '$cuisine'}, 'count':{'$sum':1}}},
     {'$project':{'_id':0, 'cuisine':'$_id.cuisine', 'count':'$count'}},
     {'$sort':{'count':-1}},
     {'$limit':5}]
Result:
[{cuisine: chinese, count: 177},
 {cuisine: coffee_shop, count: 121},
 {cuisine: japanese, count: 95},
 {cuisine: burger, count: 91},
 {cuisine: regional, count: 43}]
```

- Top 5 Fast Food

```
Pipeline:
    [{'$match': {'amenity':{'$exists':1}, 'amenity':fast_food}},
     {'$group': {'_id': {'name': '$name'}, 'count':{'$sum':1}}},
     {'$project':{'_id':0, 'cuisine':'$_id.cuisine', 'count':'$count'}},
     {'$sort':{'count':-1}},
     {'$limit':5}]
Result:
[{name: 麥當勞, count: 57},        (McDonald's)
 {name: 摩斯漢堡, count: 30},      (Mos Burger)
 {name: Subway, count: 14},
 {name: 肯德基, count: 11},        (KFC)
 {name: None, count: 10}]
```

- Top 5 Shop

```
Pipeline:
    [{'$match': {'shop':{'$exists':1 }},
     {'$group': {'_id': {'shop': '$shop'}, 'count':{'$sum':1}}},
     {'$project':{'_id':0, 'shop':'$_id.shop', 'count':'$count'}},
     {'$sort':{'count':-1}},
     {'$limit':5}]
Result:
[{name: convenience, count: 1256},
 {name: clothes, count: 374},
 {name: supermarket, count: 285},
 {name: hairdresser, count: 184},
 {name: bakery, count: 177}]
Result of city:
[{city: None, count: 1002},
 {city: 臺北市, count: 197},
 {name: 新北市, count: 57}]
```

Conclusion

During this project, the most time I spend is to process the address well. Unfortunately, most node contains no address information or the information is not enough. Thus, when I was doing the section, Additional data exploration, the results were few if I added `city`, `district` to the conditions. It is really frustrating because I expected that the address in good structure can help me to explore more useful information. For example, you can find out that there are over 1000 convenience store in Taipei city and New Taipei City. The density is really amazing. So, I really want to how many convenience stores in each district and what is the feature of that district have such number of the convenience store.

I am thinking to use existing data to explore more information about the address, so that I can get the completed information from small information. Actually, it is very easy for person to connect the street name to the district or the city it belongs to. Thus, with the information of relationship between address, I think it is quite easy to complete the information of address soon. Besides, postal code API is really helpful. I also plan to create my address API which can provide the information in English.