Data Munging for OSM data of Seoul area

Map Area - Seoul

Seoul, South Korea

> https://mapzen.com/data/metro-extracts/metro/seoul_south-korea/

I picked Seoul OSM data. Seoul is my hometown.

- Because of some conversion errors when converting to .pdf, in particular MarkDown quotations of longer Python codes distorted, so I put that Python codes in code cells, not using MarkDown quotation.
- Because of the similar reason, modification on XML tags here were unavoidable, for example the node tag is disappearing in MarkDown cell in Jupyter so it is written as < node> with a space in it
- The Python codes here will not run correctly. I quoted essential part of the codes not the whole, because separately the .py files and data files are also submitted,

Problems Encountered in the Map

The problems in the data I discovered :

1) Old 6-digit postcode are being used. It should migrate to 5-digit postcode
The government had migrated to 5 digit postcode system since 8/1/2015. The 6 digits are out-dated and not correct by now. So it will be one of most prominent items to fix.

```
Example, wrong passcode tag using 6 digits postcode. We can notice t
he timestamp is before 8/1/2015 and it makes sense.
```

> < node changeset="14730048" id="2104047042" lat="37.4824167" lon="127.0142165" timestamp="2013-01-21T07:59:27Z" uid="1182555" user="Park Munsu" version="2">
>   < tag k="name" v="서초동한신플러스타운" />
>   < tag k="building" v="apartments" />
>   < tag k="addr:postcode" v="137-040" />
> < /node>

```
Example, correct postcode tag using 5 digits scheme. The timestamp i
s after 8/1/2015.
```

> < node changeset="41459713" id="4350532506" lat="37.3888936" lon="127.122479" timestamp="2016-08-15T04:48:22Z" uid="1765666" user="Lantian" version="1">
>   < tag k="name" v="풍림아이원플러스오피스텔" />
>   < tag k="building" v="yes" />
>   < tag k="addr:city" v="성남시 (Seongnam)" />

>   < tag k="addr:street" v="서현로 (Seohyeon-ro)" />
>   < tag k="addr:postcode" v="13590" />
>   < tag k="addr:housenumber" v="170" />
> < /node>

2) Mixed data in 'addr:street' and 'addr:housenumber'
For example, following node has "마조로3길 8" in addr:street and "8" is in fact 'housenumber' which is an essential key to seek correct house address and postcode. So it has to be splitted as < tag k='addr:street' v='마조로3길' /> and < tag k='addr:housenumber' v='8' />

> < node changeset="28456622" id="3298054070" lat="37.5586549" lon="127.039998" timestamp="2015-01-28T02:15:57Z" uid="2554209" user="Coinplug" version="2">
>   < tag k="name" v="통닭가" />
>   < tag k="phone" v="+82 2-2282-2050" />
>   < tag k="addr:city" v="성동구" />
>   < tag k="addr:street" v="마조로3길 8 (행당동, 1층)" />
>   < tag k="payment:bitcoin" v="yes" />
> < /node>

```
Correct example,
```

> < node changeset="34115593" id="3750114225" lat="37.5388983" lon="127.0764576" timestamp="2015-09-19T06:34:55Z" uid="1411881" user="nice micro" version="1">
>   < tag k="addr:street" v="Achasan-ro 39-gil" />
>   < tag k="addr:housenumber" v="24" />
> < /node>

and another similar case,

> < tag k="addr:street" v="공항대로" />
> < tag k="addr:housenumber" v="38길 71" />

```
has to be refabricated as
```

> < tag k='addr:street' v='공항대로38길' />
> < tag k="addr:housenumber" v="71" />

My approach to resolve aboves will be described along with Python codes

1. Migrating old 6-digit postcode to new 5-digit one

1.1 Basic data preparation

As the first move, the Seoul OSM which I have interest in was searched and downloaded.

- **'seoul_south-korea.osm'**, 475MB osm file was downloaded from https://mapzen.com/data/metro-extracts/metro/seoul_south-korea/
- **'seoul_south-korea.sample.k50.osm'** file was generated by **'SamplingFromSeoulOSM.py'** to get sample size around 10MB, then imported into

'SamplingFromSeoulOSM.py' to get sample size around 10MB, then imported into MongoDB by **'XMLintoMongoDB.py'** and **'schema.py'** file. Many part of the codes are quoted from the 'Data Wrangling' course. I added codes to deal with 'relation' element.

As I mentioned, the 6-digits postcodes are main targets in this project but that needed external reference data to correctly pintpoint new postcodes. Two external dataset were searched after some googling - the government already published the data at https://biz.epost.go.kr and http://www.juso.go.kr. The downloaded files, a 91MB .txt file and a 12MB .xls file were imported into MongoDB.

- **'seoul_post_code_new_5digits.txt'**, 569445 rows, total Korea new postcodes with addresses was downloaded from here and imported into MongoDB by **'TXTintoMongoDB_NewPostcode.py'** file.
- **'seoul_post_code_old_6digits.xls'**, 52842 rows, total Korea old postcodes with addressses was downloaded from here and imported into MongoDB by **'EXCELintoMongoDB_OldPostcode.py'** file.

1.2 Searching the target - wrong postcodes

Following code returned 13 old 6-digit postcodes like '138-200', '138-160' and so on.

In [ ]:

```
""" Problem1_5digit_new_postcode.py """
import re
c = nodetagcoll.aggregate([{'$match':{'key':'postcode', 'type':'addr',
'value': re.compile('\d{3}-\d{3}') }},
                              {"$project": {"_id":0, "id":1,
"oldpostcode":'$value', 'newpostcode':'xxxxx'}}  ])
lc = list(c)
for doc in lc : OPCNPCcoll.insert_one(doc)
```

1.3 Gathering the clues for each OSM 'id' which has wrong postcode tag

Because there exists no such data in public which correlates an old postcode to a new one, I gathered clues which will eventually used to estimate the correct new postcodes.
After some investigation, I noticed that three fields strongly indicate a postcode - 'housename', 'dong', and 'gu', that are all essential parts forming an address in Korea. All such clues are inserted into OPCNPCcoll collection which has OSM 'id' as the identity. The table will be referenced to seek correct new postcodes in NPCcoll, the 5-digit postcodes database collection .

In [ ]:

```
""" Problem1_5digit_new_postcode.py """
pcclues = list(OPCNPCcoll.find())
for clues in pcclues :
    c = NPCcoll.aggregate([
                    {'$match': {'housename_sigungu': clues['housename']}},
                    {'$match': {'legal_dongname': clues['dong']}},
                    {'$match': {'gun': clues['gu']}},
                    ])
    lc = list(c)
    if len(lc) > 0 : OPCNPCcoll.update_one({"id": clues["id"]}, {"$set": {'n
ewpostcode':lc[0]['postcode'] }})
```

1.4 The result

Only 4 postcodes out of 13, 30% were able to get new postcodes.
Following 4 old postcodes could successfuly migrate to new postcodes by using 'housename', 'dong', 'gu' fields.
{'id': 2111822927, 'oldpostcode': '138-050', 'newpostcode': '**05645**', 'housename': '윈저하우스', 'dong': '방이동', 'gu': '송파구'},
{'id': 2111837742, 'oldpostcode': '138-110', 'newpostcode': '**05778**', 'housename': '거여2단지아파트', 'dong': '거여동', 'gu': '송파구'},
{'id': 2111844406, 'oldpostcode': '138-200', 'newpostcode': '**05801**', 'housename': '선경쉐르빌', 'dong': '문정동', 'gu': '송파구'},
{'id': 2111883319, 'oldpostcode': '138-240', 'newpostcode': '**05510**', 'housename': '더샵스타리버', 'dong': '신천동', 'gu': '송파구'}

Followings are 9 failed postcodes. For example, we don't have any further clue for '삼성일원아파트' - when I searched various comibinations like '일원삼성', '삼성일원', '일원동삼성', all returned nothing. Probably it is a mistake or the building has different official name. It would be better to take a different approach like using geo-location which will be discussed later.
{'id': 2099195503, 'oldpostcode': '135-230', 'newpostcode': 'xxxxx', 'housename': '삼성일원아파트', 'dong': '일원동', 'gu': '강남구'},
{'id': 2099308641, 'oldpostcode': '135-270', 'newpostcode': 'xxxxx', 'housename': '도곡쌍용예가클래식아파트', 'dong': '도곡동', 'gu': '강남구'},
{'id': 2099361524, 'oldpostcode': '135-100', 'newpostcode': 'xxxxx', 'housename': '두산아파트', 'dong': '청담동', 'gu': '강남구'},
{'id': 2103870093, 'oldpostcode': '137-060', 'newpostcode': 'xxxxx', 'housename': '방배현대1차아파트', 'dong': '방배동', 'gu': '서초구'},
{'id': 2104047042, 'oldpostcode': '137-040', 'newpostcode': 'xxxxx', 'housename': '서초동한신플러스타운', 'dong': '반포동', 'gu': '서초구'},
{'id': 2111825710, 'oldpostcode': '138-120', 'newpostcode': 'xxxxx', 'housename': '자연채아파트', 'dong': '마천동', 'gu': '송파구'},
{'id': 2111851586, 'oldpostcode': '138-160', 'newpostcode': 'xxxxx', 'housename': '주성드림팰리스아파트', 'dong': '가락동', 'gu': '송파구'},
{'id': 2922895302, 'oldpostcode': '100-863', 'newpostcode': 'xxxxx', 'housename': '버거킹 (Burger King)', 'dong': '충무로4가', 'gu': '중구'},
{'id': 4629321266, 'oldpostcode': '135-100', 'newpostcode': 'xxxxx', 'housename': '영동제일교회', 'dong': '청담동', 'gu': '강남구'}]

2. Reforming mixed data in 'addr:street' and 'addr:housenumber'

The street number and house number are distributed wrongly among the two tags. There are two kinds of that wrong positioning and I divided that into two approaches.

1. Extracting and migrating house number in 'addr:street' to 'addr:housenumber', for example
   < tag k="addr:street" v="Mazoro3gil **8** (haengdangdong)" />
   has to be refabricated as below because **'8'** is a 'housenumber'. (and '(haengdangdong)' is a duplicate and can be deleted)
   < tag k='addr:street' v='Mazoro3gil' />
   < tag k='addr:housenumber' v=**'8'** />
2. Extracting and migrating street number in 'addr:housenumber' to 'addr:street', for example
   < tag k="addr:street" v="Gonghangdaero" />

< tag k="addr:housenumber" v="**38gil** 71" />
has to be refabricated as below because **'38gil'** is part of 'street'
< tag k='addr:street' v='Gonghangdaero**38gil**' />
< tag k="addr:housenumber" v="71" />

The targets are picked like this codes, returning 3 items - tags of 3 OSM id - 3298054070, 4312130291, 4427829923

In [ ]:

```
"""Problem2_split_street_tag.py"""
import re
# 1st type of mixed data
nodetagcoll.aggregate([{'$match':{'key':'street', 'type':'addr', 'value': r
e.compile('.*\d+길 *\d+') }},
                            {"$project": {"_id":0, "id":1, "value":1}},
                            {'$limit':30},
                        ])
# 2st type of mixed data
nodetagcoll.aggregate([
                        {'$match':{'key':'housenumber', 'type':'addr',
'value': re.compile('.+\d길 +\d.+') }},
                        {'$project': {'_id':0, 'id':1,
'housenumber':'$value', 'type':'addr'}},
                        {'$lookup':{'localField':'id', 'from':'nodetagcoll'
'foreignField':'id','as':'othertags'}},
                        {'$unwind':{'path': '$othertags',
'preserveNullAndEmptyArrays': False}},
                        {'$match': {'othertags.key':'street'}},
                        {'$project':{'_id':0, 'id':1,
'street':'$othertags.value','housenumber':1}},
                        {'$limit':30},
                        ])
```

Then the data were splitted and allocated into correct position in correct tag

In [ ]:

```
"""Problem2_split_street_tag.py"""
correcttags = []
for d in lc :
    sraw = d['street']
    hraw = d['housenumber']
    street = ''
    r = re.findall('.+[길로][ (]', sraw)
    if r : street = r[0].replace(' ', '').replace('(', '')
    else : street = sraw
    housenumber = ''
    r = re.findall('\d+[길로]', hraw)
    if r :
        housenumber = hraw.replace(r[0], '').replace(' ', '')
        if len(street) > 0 :
            street = street + r[0]
            street.replace(' ','')
    if len(street) > 0 : correcttags.append({'id':d['id'], 'type':'addr', 'k
ey':'street', 'value': street })
    if len(housenumber) > 0 : correcttags.append({'id':d['id'], 'type':'addr
', 'key':'housenumber', 'value': housenumber })
```

The result

Every target was successfully correctified.

- First type
  {'id': 3298054070, 'type': 'addr', 'key': 'street', 'value': '마조로3길 **8** (행당동, 1층)'},
  are restructured to
  {'id': 3298054070, 'type': 'addr', 'key': 'street', 'value': '마조로3길'},
  {'id': 3298054070, 'type': 'addr', 'key': 'housenumber', 'value': '**8**'}
- Second type
  {'id': 4312130291, 'type': 'addr', 'key': 'street', 'value': '공항대로 (Gonghang-daero)'},
  {'id': 4312130291, 'type': 'addr', 'key': 'housenumber', 'value': '**38**길 71'},
  {'id': 4427829923, 'type': 'addr', 'key': 'street', 'value': '통일로'},
  {'id': 4427829923, 'type': 'addr', 'key': 'housenumber', 'value': '**87**길 21'}
  are restructured to
  {'id': 4312130291, 'type': 'addr', 'key': 'street', 'value': '공항대로**38**길'},
  {'id': 4312130291, 'type': 'addr', 'key': 'housenumber', 'value': '71'},
  {'id': 4427829923, 'type': 'addr', 'key': 'street', 'value': '통일로**87**길'},
  {'id': 4427829923, 'type': 'addr', 'key': 'housenumber', 'value': '21'}

Overview of the data

This section contains basic statistics about the dataset, the MongoDB queries used to gather them, and some additional ideas about the data in context.
File sizes

> seoul_south-korea.osm .... 474 MB
> charlotte.db .... 129 MB
> nodes.csv .... 144 MB
> nodes_tags.csv .... 0.64 MB
> ways.csv .... 4.7 MB
> ways_tags.csv .... 20 MB
> ways_nodes.cv .... 35 MB

Data overview

- Number of nodes
  43800

  > nodecoll.find().count()

- Number of chosen type of nodes, like cafes, shops etc.
  23

  > len(nodetagcoll.find({"key":"amenity", "value":"cafe"}).distinct("id"))

- Number of ways

5855

> waycoll.find().count()

- Number of unique users
  2135

> userlist = nodecoll.distinct("user")
> userset = set(userlist)
> userlist = waycoll.distinct("user")
> for u in userlist : userset.add(u)
> userlist = relationcoll.distinct("user")
> for u in userlist : userset.add(u)
> len(userset)

- Top 10 contributing users
  'maphunter36' 4949
  '전기곰돌' 2673
  'Lantian'.........................2378
  'IRTC1015' 1563
  'icemango' 1363
  'cyana'............................1321
  'Jole22'...........................1274
  '_Jibril'............................1237
  panhoong' 885
  'Albertus Liberius'........819

In [ ]:

```
import pandas as pd
df = pd.DataFrame(list(nodecoll.aggregate([{"$group":{"_id":"$user", "count
":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":20}])))
df = df.append(pd.DataFrame(list(waycoll.aggregate([{"$group":{"_id":"$user
", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":20}]))))
df = df.append(pd.DataFrame(list(relationcoll.aggregate([{"$group":{"_id":"
$user", "count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":20}]))))
df = df.groupby('_id').sum().sort_values('count' , axis=0, ascending=False)

print(df.to_dict())
```

- Number of users appearing only once (having 1 post)
  1166

In [ ]:

```
nlist = list(nodecoll.aggregate([{"$group":{"_id":"$user", "count":{"$sum":
1}}}, {"$match":{"count":1}}, { "$project": {"_id": "$_id"}}]))
valuelist = []
for dt in nlist : valuelist.append(dt["_id"])
nlist = valuelist
wlist = list(waycoll.aggregate([{"$group":{"_id":"$user", "count":{"$sum":1
}}}, {"$match":{"count":1}}, { "$project": {"_id": "$_id"}}]))
valuelist = []
```

```
for dt in wlist : valuelist.append(dt["_id"])
wlist = valuelist
rlist = list(relationcoll.aggregate([{"$group":{"_id":"$user", "count":{"$s
um":1}}}, {"$match":{"count":1}}, { "$project": {"_id": "$_id"}}]))
valuelist = []
for dt in rlist : valuelist.append(dt["_id"])
rlist = valuelist
DistictUsersofSinglePost = (set(nlist) | set(wlist) | set(rlist)) - ((set(nl
ist)&set(wlist)) | (set(wlist)&set(rlist)) | (set(rlist)&set(nlist)) )
DistictUsersofSinglePost  = list(DistictUsersofSinglePost  )
print(len(DistictUsersofSinglePost ))
```

Other ideas about the data

1) An idea to utilize [GPS coordinate to address] API which the government is running, to get the correct postcode and address
There are wrong data like mismatch in the name and postcode which lacks more clues to correctly point a postcode as we saw in the 1st problem,

> < node changeset="14730048" id="2104047042" lat="37.4824167" lon="127.0142165" timestamp="2013-01-21T07:59:27Z" uid="1182555" user="Park Munsu" version="2">
> < tag k="name" v="서초동한신플러스타운" />
> < tag k="building" v="apartments" />
> < tag k="addr:postcode" v="137-040" />
> < /node>
>
> - "서초동한신플러스타운" is not official name for that apartments and there is no such apartments in "137-040" the old postcode. We can suspect a human error.

So, I searched and discovered that there is a government-running map coordinate API which receives x, y coordinates and returns the address information. http://www.juso.go.kr/addrlink/devAddrLinkRequestGuide.do?menu=coordApi This method assumes that at least the 'lon' 'lat' values in the Node tag must be correct. It will be our last resort to correctify the data, but in the future project not now.

2) new tag key idea : new key convention like 'addr:street:en' would be required for global(English) address
For example in Asia, the participants would usually put their address in native language first and some perfectionists might put English address behind that in () as in the below example.
Sample node which added English city/street name just behind Korean inside ()

> < node changeset="46068469" id="2102619096" lat="37.4833445" lon="127.086372" timestamp="2017-02-14T04:58:23Z" uid="5315129" user="Sang Joon Kang" version="2">
> < tag k="name" v="일원본동주민센터" />
> < tag k="amenity" v="townhall" />
> < tag k="name:en" v="Irwonbon-dong Community Service Center" />
> < tag k="addr:city" v="일원동 (Irwon-dong)" />
> < tag k="addr:street" v="광평로 (Gwangpyeong-ro)" />
> < tag k="addr:postcode" v="06361" />

< tag k="addr:housenumber" v="126" />
< /node>

It would be useful to set a tag key like 'addr:street:en', 'addr:street:global' or 'addr_global:street' for the global understanding of the data. Of course, 'addr:street:global' might be adding more complexity to manipulate the third element.
There is similar one like 'name:en' for example < tag k="name:en" v="Jongno 3(sam)-ga Station gate 6" /> but no such thing like 'addr:street:en' found in http://wiki.openstreetmap.org/wiki/Key:addr

In [ ]:

```
# There are many such Eng-Kor mixed tags
import re
c = nodetagcoll.find({'key':{ '$in' : ['city','street']}, 'type':'addr', 'va
lue': re.compile('.*[a-zA-Z]+') }, {'_id':0, 'id':1, 'value':1} )
print(list(c))
```

3) An idea to add missing addr:postcode tag from the new 5digit postcode data by referencing 'addr:street' and 'addr:housenumber'
For example, node below has no tag 'addr:postcode'

> < node changeset="18074088" id="2474972664" lat="37.5915492" lon="127.0984053" timestamp="2013-09-28T09:58:30Z" uid="427807" user="samden" version="1">
> < tag k="addr:street" v="Yongmasan-ro 100-gil" />
> < tag k="addr:housenumber" v="8" />
> < /node>

In the 'seoul_post_code_new_5digits.txt', the corresponding post code was easily found using 'addr:street' and 'addr:housenumber'. The 8 in the middle indicates the house number.

> "**02188**|서울특별시||Seoul|중랑구|Jungnang-gu|||112604118468|용마산로100길|**Yongmasan-ro 100-gil**|0|**8**|0|1126010500104420015003790||경희마트|1126010500|망우동||망우제3동|0|442|01|15||"

So, < tag k="addr:postcode" v="02188" /> was able to be formed, and in such way, more data wrangling is possible for the whole dataset.

4) An idea to automatically build and add 'name:en' tag using governmental public database
Following is a well-formed example even has Japanese name

> < node changeset="45051825" id="2328266071" lat="37.3852675" lon="127.1210473" timestamp="2017-01-10T14:09:49Z" uid="1765666" user="Lantian" version="2">
> < tag k="name" v="호텔갤러리" />
> < tag k="name:en" v="Hotel Gallery" />
> < tag k="name:ja" v="ホテルギャラリー" />
> < /node>

However, following has Korean name only. Then what is the official English name for "신한은행"?

However, following has Korean name only. Then what is the official English name for "신한은행"?

> < node changeset="14583283" id="2102460016" lat="37.5248701"
> lon="127.0246497" timestamp="2013-01-09T07:56:47Z" uid="1189612"
> user="Myungsu Chae" version="1">
>   < tag k="name" v="신한은행" />
>   < tag k="amenity" v="bank" />
> < /node>

Fortunately, it turned out there is English address data in txt form, published by the Korea goverment at http://www.juso.go.kr/addrlink/addressBuildDevNew.do?menu=engj By applying the same method aboves, we can get the corresponding English name and add 'name:en' tags

Additional Data Exploration

Top 10 appearing amenities

{'count': 105, 'amenity': 'hospital'}, {'count': 50, 'amenity': 'restaurant'}, {'count': 33, 'amenity': 'bank'}, and so on

In [ ]:

```
nodetagcoll.aggregate([
        {"$match":{"key":'amenity'}},
        {"$group":{"_id":"$value", "count":{"$sum":1}}},
        {'$sort':{'count':-1}},
        {"$limit":10},
        {'$project':{'_id':0, 'amenity':'$_id', 'count':1 }}
        ])
```

Biggest religion

{'count': 21, 'amenity_religion': 'christian'}, {'count': 4, 'amenity_religion': 'buddhist'}

In [ ]:

```
nodetagcoll.aggregate([
        {"$match":{"key":'amenity', "value":"place_of_worship"}},
        {'$lookup':{'localField':'id', 'from':'nodetagcoll', 'foreignField'
:'id','as':'othertags'}},
        {'$unwind':'$othertags'},
        {'$match':{"othertags.key":'religion', 'othertags' : {'$nin' : [[]]}
}},
        {'$project':{'_id':0, 'amenity':'$key',
'religion':'$othertags.value' }},
        {"$group":{"_id":"$religion", "count":{"$sum":1}}},
        {'$sort':{'count':-1}},
        {"$limit":10},
        {'$project':{'_id':0, 'amenity_religion':'$_id', 'count':1 }}
        ])
```

Most popular cuisines

{'count': 3, 'restaurant_cuisine': 'chinese'}, {'count': 2, 'restaurant_cuisine': 'korean'}, {'count': 2, 'restaurant_cuisine': 'fried_chicken'}

In [ ]:

```
nodetagcoll.aggregate([
        {"$match":{"key":'amenity', "value":"restaurant"}},
        {'$lookup':{'localField':'id', 'from':'nodetagcoll', 'foreignField'
:'id','as':'othertags'}},
        {'$unwind':'$othertags'},
        {'$match':{"othertags.key":'cuisine', 'othertags' : {'$nin' : [[]]}}
},
        {'$project':{'_id':0, 'amenity':'$key',
'cuisine':'$othertags.value' }},
        {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
        {'$sort':{'count':-1}},
        {"$limit":10},
        {'$project':{'_id':0, 'restaurant_cuisine':'$_id', 'count':1 }}
        ])
```

Conclusion

To retrieve the correct new 5-digit postcode, I gathered and utilized public data, but it was not that successful, 4 successful migrations to new postcode out of 13, 30%. It is because it turned out the clues are limited to clearly point the house address, for example the house name had various versions and were not easy to assert the equality between two different versions.
However, I got to know there exists a public address API service based on the geo-location -lon and lat- , http://www.juso.go.kr/addrlink/devAddrLinkRequestGuide.do?menu=coordApi , which people might be able to use to get the correct postcode rather than my tedious approach.

During the external data preparation phase - importing public address data in TXT and Exel files each, I got to know why people are using MongoDB for this sort of work. MongoDB is well-adapted to blunt importing (meaning not much considering of the table schema) and manipulating the data and extracting some meaning as fast as the programmer can. Even though some people say the syntax is rather complicated, once got used to it, the user can deal with the data more conveniently than SQL, which is too strict.

There were several MongoDB behaviors I cannot fully understand it is by design or a bug, for example 1) when pouring OSM into MongoDB, an 'id' field turned into 'string' not 'integer' though other 'id' fields in other collections had correct 'integer', under the same schema and process, 2) when I joined($lookup) two collections, though there clearly were common key values, the result was empty. The result was correct only when I reversed the direction of the join. It took considerable time to work around the issues and still not clearly understood. I hope to raise this issue if I can form easily reproducible and simple dataset.

In [ ]: