OpenStreetMap Sample Project

Data Wrangling with MongoDB

Stefan Erfurth

Map Area: Area around Mayen (area where I originally come from ☺), Rhineland-Palatinate, Germany

1. Problems Encountered in the Map

1. Documents have keys or tags “type”
2. Tags/keys with different names
3. Key’s containing ‘.’ when inserting data into MongoDB
4. Postcode check/ double naming of cities

1. **Documents have keys or tags “type”**
   One data validation check was to check which different values the tag ‘type’ had. Due to the code I used from lesson 6, I expected to get only ‘way’ and ‘node’ as a result but there were also other values, i.e. [u'node', u'enforcement', u'busbar', u'Rosskastanie', u'Schrank', u'gas', u'box', u'Eiche', u'TMC', u'way', u'broad_leaved', u'multipolygon'].
   Thus either the tag or the key ‘type’ existed in the extract. I therefore included an if-clause in the ‘shape elements’ list which checks for those keys/tags and for those a key ‘osm_type’ is added.
   This affected only 0.1% of the cases which is thus a really minor issue.

2. **Tags/keys with different names:**
   Before passing the json file to mongoDB, I checked which different tags/keys have been in the original data and were therefore written into the json file as keys of the single dictionaries. I then went through these 738 tags/keys manually to check for potential duplicates w.r.t. meaning of the tags/keys. I looked at examples for the respective duplicates to see whether those are really duplicates and therefore could identify some which haven’t been actual duplicates even though they looked like this in the beginning (e.g. ‘bic’ which is the “Bank Identifier Code” used in the EU and not an abbreviation for ‘bicycle’ which is included as well).
   Nevertheless, I found the following duplicates which I replaced by the name of the other. I therefore always replaced that key/tag which occurred less often in the sample data:

a. ‘acc’ -> ‘access’
b. ‘alternative_name’ -> to ‘alt_name’
c. contact: besides eliminating duplicates (e.g. ‘contact:email’ and ‘email’), I created here the same structure as for ‘created’: all contact information is saved in a sub-dictionary even though this is not a pre-requisite as of osm-Wiki: http://wiki.openstreetmap.org/wiki/Annotations
d. ‘FIXME’->’fixme’
e. ‘Öffnungszeiten’ -> ‘opening_hours’
f. ‘Öffungszeiten’ -> ‘opening_hours’

PythonCode:

```
tag_replacement = {'acc': 'access', 'alternative_name': 'alt_name', 'FIXME': 'fixme', 'Öffnungszeiten':
'opening_hours', 'Öffungszeiten': 'opening_hours', 'step.condition': 'step:condition', 'surface.material':
'surface:material'}
def replaceDuplicateTags(lstOfDicts, lstTagMapping):
    #1. loop through all elements in the list
    newLstOfDicts = []
    counterUpdates = 0
    for el in lstOfDicts:
        CurrEl = el
        #2. loop through all keys in the dictionary
        for key in CurrEl:
            #3. if the key is in the list lstTagMapping
            if key in lstTagMapping:
                #4. create the entry with the new name and delete the old one
                counterUpdates +=1
                CurrEl[key] = CurrEl.pop(key)
        newLstOfDicts.append(CurrEl)
    return newLstOfDicts, counterUpdates
#96 changes made

contact_list = {'contact:email': 'email', 'contact:fax': 'fax', 'contact:google_plus': 'google_plus',
'contact:mobile': 'mobile', 'contact:office': 'office', 'contact:phone': 'phone', 'contact:website': 'website',
'email': 'email', 'fax': 'fax', 'phone': 'phone'}
def updateContacts(lstOfDicts, lstContact):
    newLstOfDicts = []
    counterUpdates = 0
    #1. loop through all elements in the list
    for el in lstOfDicts:
        CurrEl = el
        lstDelCurrEl = []
        contactAdded = False
        #2. loop through all keys in the dictionary
        for key in el:
            #3. if the key is in the list contact
            if key in lstContact:
                if not(contactAdded):
                    counterUpdates +=1
                    contactAdded = True
                    #4. if contact is not already a key in the dictionary, create the key with an empty dictionary
                    contact = {}
                #5. create the entry in the dict 'contact' and delete the old entry
                contact[lstContact[key]] = CurrEl[key]
        #6. delete all original keys, append to new lst of dicts and add contacts if included
        for k in lstDelCurrEl:
            del CurrEl[key]
        if contactAdded:
            CurrEl['contact'] = contact
        newLstOfDicts.append(CurrEl)
    return newLstOfDicts, counterUpdates
#300 changes made
```

As less than 0.1% of the data was changed, the effort in this case was actually not completely worth the cleaning.

3. **Key's containing '.' when inserting data into MongoDB**
   The first try of inserting the data from OMS into MongoDB failed due to the reason that there existed two keys/tags which had a '.' in their name. This issue was fixed by including those into the tag_replacement list within item 1.2 and replaced them with respective names.

4. **Postcode check/ double naming of cities**
   There exist postcodes which are used for different cities. This is partially driven by the fact that a postcode is valid for several villages but partially also driven by the fact that there are several potential ways to write the postal address of a city which is the criteria for city as of the OSM wiki: http://wiki.openstreetmap.org/wiki/Key:addr

   The following example (cities with postcode 56745) illustrates both cases:

City	Postcode	Comment
Bell	56745	Correct postcode, shared with other villages
Rieden (Eifel)	56745	Correct postcode and correct name to identify village unambiguously in Germany/Austria/Switzerland as other villages/towns with name Rieden exist, valid postal address; commonly used
Rieden	56745	Correct postcode and correct official name
Volkesfeld	56745	Correct postcode, shared with other villages
Weibern	56745	Correct postcode, shared with other villages

   Depending on the way one wants to use the data (e.g. more detailed analysis per postal code/ city) one name should be the standard. In this case, other issues pop up: What to do with villages which have been incorporated by another city/village? Legally they are on the one hand side part of the other city/village but they are to some degree also legally independent and often geographically not directly connected with the "father town".

   Having a look at cities which have several postcodes, only 'Neuwied' and 'Koblenz' can be found which coincides with the number of cities having several postcodes in the selected map area.

2. Data Overview

2.1 First analysis from python:

of elements: 593,561

Before solving nodes problem (see part 1):

# of nodes: 529,819
# of ways: 63,669

After solving node problem (see part 1):

# of nodes: 529,885
# of ways: 63,676

File size:

- MYK.osm: 116 MB
- MYK.osm.json: 126 MB

2.2 Analysis within MongoDB

# of elements: 593,561 (same as in python)
db.MYK.count()

# of nodes: 529,885 (same as in python)
db.MYK.find({'type':'node'}).count()

# of ways: 63,676 (same as in python)
db.MYK.find({'type':'way'}).count()

# of cities: 121
```
pipeline = [{'$match': {'address.city': {'$exists': True}}},
        {'$group': {'_id':{'city': '$address.city'}}}]
len(list(db.MYK.aggregate(pipeline)))
```

# of postcodes: 46
```
pipeline = [{'$match': {'address.postcode': {'$exists': True}}},
        {'$group': {'_id':{'postcode': '$address.postcode'}}}]
len(list(db.MYK.aggregate(pipeline)))
```

# of postcodes appearing in more than one city: 22 (used for 1.5)
```
pipeline = [{'$match': {'address.city': {'$exists': True},
                'address.postcode': {'$exists': True}}},
        {'$group': {'_id':{'city': '$address.city', 'postcode': '$address.postcode'}}},
        {'$group': {'_id': '$_id.postcode',
                'count': {'$sum': 1}}},
        {'$match': {'count': {'$gt': 1}}}]
len(list(db.MYK.aggregate(pipeline)))
```

# of cities having more than one postcode: 2
```
pipeline = [{'$match': {'address.city': {'$exists': True},
                'address.postcode': {'$exists': True}}},
        {'$group': {'_id':{'city': '$address.city', 'postcode': '$address.postcode'}}},
        {'$group': {'_id': '$_id.city',
                'count': {'$sum': 1}}},
```

{'$match': {'count': {'$gt': 1}}}]
len(list(db.MYK.aggregate(pipeline)))

3. Additional Ideas

1. **Which contact information is given most often?**
   As mentioned in 1.2, I included all contact information in a group of fields/ a dictionary. In there, I check what kind of contact information is given how often. For a further project this could give an indication which data is missing most often.

   In 328 cases at least one type of contact information is given.
   pipeline = [{'$match': {'contact': {'$exists': True}}}]

   In 317 cases, phone is given as a contact detail, followed by email in 80 cases and fax in 70 cases. Complete list:
   email 80, fax 70, google_plus 2, mobile 1, office 1, phone 317, website 18
   for ContactMode in ['email', 'fax', 'google_plus', 'mobile', 'office', 'phone', 'website']:
   pipeline = [{'$match': {'contact.' + ContactMode: {'$exists': True}}}]
   print ContactMode, len(list(db.MYK.aggregate(pipeline)))

   Assuming the data is to some degree complete, one could even think of indicators which represent how digital certain areas in an area/ a country / a region are. Here I could think of the ratio of (# of fax numbers provided/ # of e-mail address provided).

2. **How often are names in other languages stored?**
   Besides one given name which is probably provided in German, for some objects there is also a name in the data in another language with Spanish and French leading besides names are labeled as German: name:de 27, name:es 21, name:fr 21, name:en 4, name:it 2, name:nl 2, name:ru 2, name:sk 2, name:hu 1
   for Name in ['name', 'name:de', 'name:en', 'name:es', 'name:fr', 'name:hu', 'name:it', 'name:nl', 'name:ru', 'name:sk']:
   pipeline = [{'$match': {Name: {'$exists': True}}}]
   print Name, len(list(db.MYK.aggregate(pipeline)))

   In the current situation in Europe, especially in German, with many refugees coming from mainly Syria, Afghanistan and Albania, a project could be to further add names in different languages. When the refugees come to Europe, they are in general not directly allowed to work and often do not have a task and cannot do anything in their new environment. In a project where they should enter the names together with locals, they would get in contact with locals, are to some degree educated in something new and ensure that other refugees would know where to find what in their new environment. Maybe it would make more sense to update entries in googlemapes instead, as this is in wide use of people from all over the world, while having the disadvantage that the data would be owned by a company.

4. Conclusion

Some issues have been encountered in the map which partially have been solved. Other would still need to be solved depending on the analysis one further would further want to conduct. A potential cleaning would be setting city names all to the same as described in 1.4. In general one can see that the data quality is in a good shape as the issues identified covered only a small percentage of the data.

Due to the general good shape, further analysis and further usage of the data could be made as described in the additional ideas area.