Project: Wrangle OpenStreetMap Data

Data Wrangling with MongoDB Jocelyn Moreau

Map Area: Finistere sud, Bretagne, France

https://export.hotosm.org/en/v3/exports/382e36c5-7823-4848-b664-6fcdc6b533ee

The OSM is 50,1Mo.

In this project, we are going to address the following points:

1. Prepare and sample the OSM file
   - Number of tags (mapparser.py)
   - Analyse the problematic tags (tags.py)
   - Audit the file (audit.py)
2. Import in MongoDB and requested analysis
3. Additional Statistics and Ideas
   - Contributor statistics
   - Additional data exploration using MongoDB
4. Conclusion

1. Prepare and sample the OSM file

For this project, I decided to work on a region of France, called Finistere sud, in Bretagne, France, a nice area for vacation! The full map is available as france-finistere-sud_export.osm (Size of 50,1Mo, so compliant with the >50Mo size) in the current Github repository. I extracted a sample using the sample_file.py script (see cell below). The result is saved in france-finistere-sud_export_sample.osm. This is the file I will use for the next steps.

In [1]:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# code stored in sample_file.py

import xml.etree.ElementTree as ET  # Use cElementTree or lxml if too slow

OSM_FILE = "france-finistere-sud_export.osm"  # Replace this with your osm
file
SAMPLE_FILE = "france-finistere-sud_export_sample.osm"

k = 10 # Parameter: take every k-th top level element

def get_element(osm_file, tags=('node', 'way', 'relation')):
    """Yield element if it is the right type of tag
    Reference:
    http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-fi
le-generated-via-xml-etree-elementtree-in-python
    """
    context = iter(ET.iterparse(osm_file, events=('start', 'end')))
    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()


with open(SAMPLE_FILE, 'w') as output:
    output.write('<?xml version="1.0" encoding="UTF-8"?>\n')
    output.write('<osm>\n  ')
    # Write every kth top level element
    for i, element in enumerate(get_element(OSM_FILE)):
        if i % k == 0:
            output.write(str(ET.tostring(element, encoding='utf-8')))

    output.write('</osm>')
    print("sample done")
```

```
sample done
```

1.1 Number of tags (mapparser.py)

Counting the number of nodes for each file (using mapparser.py script)

In [2]:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Map parser to count the number of tags of the OSMFILE
mapparser.py
"""

import xml.etree.cElementTree as ET
import pprint

def count_tags(filename):
    tags = {}
    for event, elem in ET.iterparse(filename):
        if elem.tag in tags:
            tags[elem.tag] += 1
        else:
            tags[elem.tag] = 1
    return tags

pprint.pprint(count_tags(OSM_FILE))
```

```
{'member': 49,
 'meta': 1,
 'nd': 275503,
 'node': 216346,
 'note': 1,
 'osm': 1,
 'relation': 24,
 'tag': 74529,
 'way': 34610}
```

In [3]:

```
# Count the tags for the sample file
pprint.pprint(count_tags(SAMPLE_FILE))
```

```
{'member': 4,
 'nd': 27548,
 'node': 21635,
 'osm': 1,
 'relation': 2,
 'tag': 7450,
 'way': 3461}
```

In [4]:

```python
#script used 2-tags.py
import re

lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

def key_type(element, keys):
    if element.tag == "tag":
        for tag in element.iter('tag'):
            k = tag.get('k')
            if lower.search(k):
                keys['lower'] += 1
            elif lower_colon.search(k):
                keys['lower_colon'] += 1
            elif problemchars.search(k):
                keys['problemchars'] += 1
            else:
                keys['other'] += 1

    return keys

def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}

    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)

    return keys

keys = process_map(OSM_FILE)
pprint.pprint(keys)
```

```
{'lower': 74018, 'lower_colon': 463, 'other': 42, 'problemchars': 6}
```

Legend:

- "lower", for tags that contain only lowercase letters and are valid,
- "lower_colon", for otherwise valid tags with a colon in their names,
- "problemchars", for tags with problematic characters, and
- "other", for other tags that do not fall into the other three categories.

This first analysis shows us that there is:

- 74018 tags having no upper case, which means that will have to add an uppercase for the first letter of the tag in the next step.
- 463 tags with a colon (a short look shows these are web addresses)
- 6 problematic characters in the map chosen

- 42 others, which seem to be fine (they are tags for mentioning the source of the information, SIREN (French unique reference number for company) ref. ...

  ### 1.3 Audit the file (audit.py) After auditing the sample file with the audit.py script, I could not detect any error... so I decided to run the audit on the complete OSM file.

In [5]:

```
#audit.py
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint

street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
expected = ["Concarneau", "Rue", "Impasse", "Avenue", "Pont", "Boulevard",
"Porte", "Route", "Chemin"] #expected names in the dataset
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

#Array to update with the errors encountered in the file
mapping = {"bretagne": "Bretagne",
           "Av.": "Avenue",
           "pont": "Pont",
           "rue": "Rue",
           "Bd" : "Boulevard",
           "bd" : "Boulevard",
           "Pt" : "Pont",
           "Rte": "Route",
           "place": "Place",
           "quai": "Quai",
           "chemin": "Chemin"
           }

# Search string for the special characters and compare the result to the
expected list. If not inside, then add it to the street_type array.
def audit_street(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)

# Check streetname
def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

# Audit function
def audit(osmfile):
    osm_file = open(osmfile, encoding="utf8")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street(street_types, tag.attrib['v'])

    return street_types

 # reformat string to first letter capital, except for abreviation all in
```

```python
# reformat string to first letter capital, except for abreviation all in
capital
def string_case(s):
    if s.isupper():
        return s
    else:
        return s.title()

# update name function
def update_name(name, mapping):
    name = name.split(' ')
    for i in range(len(name)):
        if name[i] in mapping:
            name[i] = mapping[name[i]]
            #reformat
            name[i] = string_case(name[i])
        else:
            name[i] = string_case(name[i])

    name = ' '.join(name)


    return name
print("Get the list of existing name")
print("##############################")
pprint.pprint(dict(audit(OSM_FILE))) # print the existing names

print()
print("List of items with proposed correction:")

st_types = audit(OSM_FILE)

# print the updated names
for street_type, ways in st_types.items():
    for name in ways:
        better_name = update_name(name, mapping)
        print (name, "=>", better_name)
```

```
Get the list of existing name
##############################
{'Argant': {'Stang Argant'},
 'Berthou': {'Rue Joseph Berthou'},
 'Bienvenue': {'Rue Fulgence Bienvenue'},
 'Bras': {'Hent Mort Bras'},
 'Carnot': {'Quai Carnot'},
 'Colguen': {'Rue de Colguen'},
 'Courcy': {'rue de Courcy', 'Rue de Courcy', 'Rue Courcy'},
 'Croix': {'Quai de la Croix'},
 'Césaire': {'Rue Emile Césaire', 'Rue Aimé Césaire'},
 'Duguesclin': {'Place Duguesclin'},
 'Duquesne': {'Rue Duquesne'},
 'Flaubert': {'Rue Gustave Flaubert'},
 'Foch': {'Rue Maréchal Foch'},
 'Gaulle': {'Place Général de Gaulle'},
 'Guillou': {'Boulevard Alfred Guillou'},
 'Guéguin': {'Avenue Pierre Guéguin'},
 'Kerdavid': {'Kerdavid', 'Route de Kerdavid'},
 'Lanriec': {'Rue de Lanriec'},
 'Madec': {'Rue René Madec'},
 'Malakoff': {'Malakoff'},
 'Morvan': {'Rue du Général Morvan'},
 'Neuve': {'Rue Neuve'},
 'Novembre': {'Rue du 11 Novembre'},
 'Pins': {'Allée des Pins'},
 'Ruat-Vraz': {'Impasse de Ruat-Vraz'},
 'Sables-Blancs': {'Rue des Sables-Blancs'},
 'Schoelcher': {'Rue Victor Schoelcher'},
 'Vert': {'Rue du Poteau Vert'},
 'Vidie': {'Rue Lucien Vidie'},
 'gare': {'rue de la gare'},
 "l'Alma": {"Rue de l'Alma"},
 "l'Océan": {"Avenue de l'Océan"},
 "l'Église": {"Rue de l'Église", "Place de l'Église"},
 'neuve': {'rue neuve'},
 'Écoles': {'Rue des Écoles'}}

List of items with proposed correction:
Rue des Écoles => Rue Des Écoles
Avenue Pierre Guéguin => Avenue Pierre Guéguin
Hent Mort Bras => Hent Mort Bras
Impasse de Ruat-Vraz => Impasse De Ruat-Vraz
Allée des Pins => Allée Des Pins
Kerdavid => Kerdavid
Route de Kerdavid => Route De Kerdavid
Avenue de l'Océan => Avenue De L'Océan
rue neuve => Rue Neuve
Rue de l'Église => Rue De L'Église
Place de l'Église => Place De L'Église
Rue Fulgence Bienvenue => Rue Fulgence Bienvenue
Rue du 11 Novembre => Rue Du 11 Novembre
Rue des Sables-Blancs => Rue Des Sables-Blancs
Boulevard Alfred Guillou => Boulevard Alfred Guillou
Quai de la Croix => Quai De La Croix
Rue Duquesne => Rue Duquesne
rue de Courcy => Rue De Courcy
Rue de Courcy => Rue De Courcy
Rue Courcy => Rue Courcy
Rue Maréchal Foch => Rue Maréchal Foch
Malakoff => Malakoff
Rue de l'Alma => Rue De L'Alma
Rue du Général Morvan => Rue Du Général Morvan
Place Général de Gaulle => Place Général De Gaulle
Place Duguesclin => Place Duguesclin
Rue Joseph Berthou => Rue Joseph Berthou
Quai Carnot => Quai Carnot
Rue de Lanriec => Rue De Lanriec
rue de la gare => Rue De La Gare
Rue Gustave Flaubert => Rue Gustave Flaubert
Rue du Poteau Vert => Rue Du Poteau Vert
Rue Neuve => Rue Neuve
Stang Argant => Stang Argant
Rue de Colguen => Rue De Colguen
Rue Lucien Vidie => Rue Lucien Vidie
Rue René Madec => Rue René Madec
Rue Emile Césaire => Rue Emile Césaire
Rue Aimé Césaire => Rue Aimé Césaire
Rue Victor Schoelcher => Rue Victor Schoelcher
```

The data has a very good quality, very few mistake were detected.

Lower case mispelling:

Lower case mispelling:

- rue -> Rue

I updated the audit.py dictionnary to include the detected mistake and ran the script again to correct all these mispelling.

Abbreviations: There was no abbreviation in the map chosen (either a student from Udacity applied this training to correct the mistake, or Openstreetmap implemented a script to correct it) --> nothing to do here.

2. Import in MongoDB

2.1 Transform the xml into json format ()

I used the following data_json.py script that turns the xml into json, cleaning the mistakes detected using the previously used functions.

In [6]:

```
# -*- coding: utf-8 -*-
# Export to JSON for MONGO DB---------------------------------------------------
------------------------------
# the corrected data are shaped in a dictionary in order to be saved into a
JSON file in order to be imported by MongoDB later on.
# The following operations are performed:
# •only 2 types of top level tags: "node" and "way" are processed
# •all attributes of "node" and "way" are turned into regular key/value pai
rs
# •some attributes "version", "changeset", "timestamp", "user", "uid" are a
dded under a key "created"
# •attributes for latitude and longitude are added to a "pos" array,
# •address related items are added to the tag "address"
# •other second level tag "k" are added to the field "others"
# script 4-data.py
import codecs
import json
import xml.etree.cElementTree as ET
from bson import json_util

# List of structure fields for created for the json file
CREATED = [ "version", "changeset", "timestamp", "user", "uid"]
#OSM_FILE = "finistere-sud-france_export.osm"

def shape_element(element):
# Create a JSON file as a preparation step for MongoDB DataBase
# Shape the XML file into a JSON alike structure
    node = {}
    # process only 2 types of top level tags: "node" and "way"
    if element.tag == "node" or element.tag == "way" :
        # "type"
        node['type'] = element.tag
        # "id"
        if 'id' in element.attrib:
            node['id'] = element.get('id')
        # "visible"
        if 'visible' in element.attrib:
            node['visible'] = "true"
            node['visible'] = element.get('visible')
        # "pos"
        if 'lat' in element.attrib and 'lon' in element.attrib:
            node['pos'] = [0,0]
            node['pos'][0] = float(element.get('lat'))
            node['pos'][1] = float(element.get('lon'))
        # "created"
        for key in element.attrib:
            value = element.attrib[key]
            if "created" not in node.keys():
                    node["created"] = {}
            if key in CREATED:
                node["created"][key] = value
        # 2nd level "address"
        for tag in element.iter("tag"):
            key = tag.attrib['k']
            value = tag.attrib['v']
            # Apply correction
            if is_street_name(tag): # if the key value is "addr:street"
                m = street_type_re.search(value)
                if m:
                    street_type = m.group() # Boulevard
                    if street_type not in mapping:
                        value = update_name(value, mapping)
        # 2nd level "others"
            elif not problemchars.match(key):
                if "others" not in node.keys():
                    node["others"] = {}
                node["others"][key] = value
        # 2nd level "way"
        for tag in element.iter("nd"):
            if "node_refs" not in node.keys():
                node["node_refs"] = []
            node["node_refs"].append(tag.attrib['ref'])
        return node
    else:
        return None

def process_map(file_in, pretty = False):
    file_out = "{0}.json".format(file_in)
    with open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                if pretty:
                    fo.write(json.dumps(el, indent=2, default=json_util.defa
ult)+"\n")
                else:
                    fo.write(json.dumps(el, default=json_util.default) + "\n
")

print ("# JSON file for import into Mongo DB")
process_map(OSM_FILE, True)
print ('--> Done!')
```

```
# JSON file for import into Mongo DB
--> Done!
```

2.2 Import the generated json into MongoDB

2.2 Import the generated json into MongoDB

I imported the json file (70,1 Mo) size into a MongoDB database (called francefinisteresud), using the mongoimport library. As a GUI for the database requests, I will use MongoBooster and copy paste below the result of the requests.

2.3 Data overview

Let's check the data we imported:

Number of documents

> db.francefinisteresud.find({}).count()

-> 250950

Number of unique users

> db.francefinisteresud.distinct('created.user').length

-> 90 differents users

Number of nodes

> db.francefinisteresud.find({"type":"node"}).count()

-> 216346 results (which corresponds to the number of nodes we counted in the xml file)

Number of ways

> db.francefinisteresud.find({"type":"way"}).count()

-> 34604 (instead of 34610: 6 ways were not imported, error message was as following: key ref:clochers.org must not contain '.', I will not try to fix this error for now)

Number of schools:

> db.francefinisteresud.find({'others.amenity': "school"}).count() --> 10

Number of restaurants

> db.francefinisteresud.find({'others.amenity': "restaurant"}).count() --> 21

Additional Statistics and Ideas

Top 10 users

> db.francefinisteresud.aggregate([{' group': {'_id': ' created.user', 'count': {' sum' : 1}}}, {' sort': {'count' : -1}}, {'$limit': 10}])

The top ten users is as follow:

id	count
eric_G	82709
Michelwald	54536
isnogoud	49702
osmmaker	21470
Super-Map	11550
PierenBot	10877
jo2929	8962
luiswoo	2358
lcroc	2165
Fifi Fofo	1518

count of each different amenity referenced:

> db.francefinisteresud.aggregate([{' group': {'_id': ' others.amenity', 'count': {' sum' : 1}}}, {' sort': {'count' : -1}}])

-> here is the result: (the first result with null is the total number of nodes without tag amenities)

_id	count
null	250854
place_of_worship	23
restaurant	21
school	10
pharmacy	6
bank	4
toilets	5
townhall	3
fuel	3
cafe	3
fast_food	3
bar	2
fire_station	2
public_building	2

_id	count
car_wash	1
parking	1
community_centre	1
post_office	1
cinema	1
marketplace	1
pub	1
police	1
recycling	1

Additional ideas

As the xml format is flexible we could think of adding many more information, linked to the different amenities listed in the map. For restaurant, for example, users could add reviews of the food, pictures, comments (the way google is doing it). Of courses, this would imply a good moderator system, not to publish incorrect data and make sure that the comments are fair for example, that the picture are the one showing the restaurant...

Another idea could be to implement a layer, on top of the map, that shows with different colors the level of completeness of the map, helping willing users to work on the porrly documented areas first. For the area I analysed with coordinate between -3,99272 and -3,72681 (middle point being -3,834) and 47,7684 and 47,94464 (middle point 47,85), I could divide the area in 4 squares and count per squares the number of nodes.

Zone 1 (lower left area)

> db.francefinisteresud.find({ and: [{'pos.0': { lt: 47.85}, {'pos.1': {$lte: -3.834}}]}).count() -> 38449

Zone 2 (upper right area)

> db.francefinisteresud.find({ and: [{'pos.0': { gte: 47.85}, {'pos.1': {$gt: -3.834}}]}).count() -> 33351

Zone 3 (lower right area)

> db.francefinisteresud.find({ and: [{'pos.0': { lt: 47.85}, {'pos.1': {$gt: -3.834}}]}).count() -> 22044

Zone 4 (lower right area)

> db.francefinisteresud.find({ and: [{'pos.0': { gte: 47.85}, {'pos.1': {$lte: -3.834}}]}).count() -> 122502

The total matches the total number of nodes of 216346. Here we can see that even on a small area like this one, the number of nodes per area is huge like this one, the number of nodes per area is huge.

Conclusion

The area audited, cleaned and analysed, showed a very good level of data integrity. Very few mistakes were found. One easy improvment on openstreetmap side, would be to automatically tranform the first letter of a street name to an upper case. This project allowed me to learn a new database language (MongoDB) which is completly different from SQL, but allowing to store data in a much more flexible way, although it can become very messy, very quickly, if no standards are followed.

References

- All python scripts used to audit, clean and analyse the data come from the case study provided in the previous lesson
- the mongoDB queries were built using https://docs.mongodb.com
- queries were executed using Mongoboost software