DATA WRANGLING WITH MONGODB

Map Area: Khmelnytskyi raion (a district) of the Khmelnytskyi Oblast in western Ukraine.

- https://www.openstreetmap.org/relation/1739099
- http://overpass-api.de/api/map?bbox=26.5306,49.2906,27.3257,49.6103
- https://en.wikipedia.org/wiki/Khmelnytskyi_Raion

I live in this place, and I want to play with requests to this database. I want to contribute to improving the map of my hometown at OpenStreetMap.org.

Preliminary analysis

For the preliminary analysis of the data, I will create a dictionary that includes all records of osm-file where the type of elements is the tag. The key of a tag will be the key of the dictionary, and the value will be a set that will include all the values of tags.

The dictionary has 442 entries that I'm going to explore.

- 30 tags converted into integers
- Two tags converted into float numbers
- 133 tags rejected because they are names of objects in languages that I don't know.
- 163 tags rejected because they have only one value.
- 33 tags rejected because they have values that equal to values from osm-wiki and contains nothing superfluous.

81 tag left to handle manually

Solving problems in the tags

tag 'addr:housenumber'

```
list(sorted(content['addr:housenumber']))[319:325]
```

```
['29/1', '2A', '2a', '3', '3/1', '3/2']
```

House numbers include digits and letters. They have large and small letters and also have Latin letters instead of Cyrillic.

```
def addr_housenumber(item):
    """ The function converts house number to uppercase
        and replaces Latin letter 'A' to Cyrillic letter 'А'.

    Args:
        item (str): house number

    Returns:
        str: fixed house number
    """
    return item.upper().replace('A', 'А')
```

tags 'name', 'description' and 'addr:street'

```
[item for item in content['addr:street'] if '.' in item]
```

```
['вул.Лісогринівецька', 'пр.Миру 102/2\n', 'пр. Миру']
```

Street names can contain abbreviations and special marks.

```python
def addr_street(item):
    """ The function deletes separator ( \n, \t) from a name of a street
        and replaces Cyrillic abbreviations to the full name.

    Args:
        item (str): name of a street

    Returns:
        str: fixed name of a street
    """
    item = ' вулиця '.join(item.split( 'вул.'))
    item = ' проспект '.join(item.split( 'пр.'))
    return ' '.join(item.split())
```

tag 'addr:city'

```python
content['addr:city']
```

```
{'Гвардійське',
 'Хмельницкий',
 'Хмельницький',
 'Чорний Острів',
 'ст. Богданівці',
 'хмельницкий',
 'хмельницький'}
```

The name of the two cities recorded twice. The first time the name starts with a capital letter a second time with a small letter. And one name contains abbreviation.

```python
def addr_city(item):
    """ The function capitalizes Cyrillic letter 'х' in the start of the word
        and replaces Cyrillic abbreviations to the full name.

    Args:
        item (str): name of a city

    Returns:
        str: fixed name of a city
    """
    if item.startswith( 'ст.'):
        item = 'станція' + item[3:]
    elif item.startswith( 'х'):
        item = 'Х' + item[1:]
    return item
```

tag 'phone' and 'phone_1'

```python
lens = set(range(100))
for phone in content['phone']:
    if len(phone) in lens:
        lens.remove( len(phone))
        print(phone)
```

```
+380382651217
(0382) 777-832
+38 0382 74-20-87
```

```
+38 0382 784767
0382 702030, 067 3831328
+30382701010
+38 (03822) 3-2241
+38 (050) 436-19-45
0382704903
0 (97) 110 50 69
719700
64-74-84
```

```
# What symbols did use to enter phone numbers?
print(list(sorted(set(i for item in content['phone'] for i in item))))
```

```
[' ', '(', ')', '+', ',', '-', '/', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

```
# phone nubera with '/' and ','
[item for item in content['phone'] if ('/' in item) or (',' in item)]
```

```
['0382 702030, 067 3831328',
 '65-74-74 / 65-10-91',
 '067-311-81-95 / 71-86-45',
 '067-327-40-67 / 65-54-93']
```

Some items have delimiters and contain two phone numbers. Phone numbers should be in international (ITU-T E.164) format.

```
def format_phone(phone):
    """ The function converts phone number to international (ITU-T E.164) format
        following the pattern '+<country code> <area code> <local number>'
        http://wiki.openstreetmap.org/wiki/Elements

    Args:
        item (str): numbers of phones with delimiters ',' or '/'

    Returns:
        list of str: corrected phone number
    """
    phones = phone.replace('/', ',').split(',')
    for i in range(len(phones)):
        item = ''.join(list(reversed([x for x in phones[i] if x.isdigit()])))
        if len(item) > 7:
            item = item[:7] + ' ' + item[7:]
        if len(item) > 11:
            item = item[:11] + ' ' + item[11:]
        if len(item) > 12:
            item = item + '+'
        phones[i] = item[::-1]
    return phones
```

tag 'addr:postcode'

```
[item for item in content['addr:postcode'] if len(item) != 5]
```

```
['29000-29499', '2900']
```

One postcode has the error less than five characters. Another postcode is the range of codes entered in one field. I will not import these postcodes.

Now I ready to make json-file and export this file in MongoDB

Data Overview

File sizes

```
!stat --printf='%s bytes  -  %n\n' khmelnytskyi*
```

```
110144963 bytes  -  khmelnytskyi_region.json
74945880 bytes  -  khmelnytskyi_region.osm
```

About the dataset

Number of documents

```
db.khmelnytskyi.count()
```

```
388520
```

Number of documents by type

```
aggregate([
    {'$group':{'_id':'$type', 'count':{'$sum': 1}}},
    {'$sort':{'_id': 1}}
])
```

```
[{'_id': 'node', 'count': 349488},
 {'_id': 'relation', 'count': 460},
 {'_id': 'way', 'count': 38572}]
```

Number of unique users

```
len(db.khmelnytskyi.distinct( "user"))
```

```
263
```

Top 5 contributing user

```
aggregate([
    {'$group':{'_id':'$user', 'count':{'$sum': 1}}},
    {'$sort':{'count': -1}},
    {'$limit': 5}
])
```

```
[{'_id': 'georg troyan', 'count': 247803},
 {'_id': 'teologovsan', 'count': 37627},
 {'_id': 'Vulpes-Vulpeos', 'count': 22903},
 {'_id': 'dexter_khm', 'count': 18667},
 {'_id': 'InnerIn', 'count': 15889}]
```

Number of users appearing only once (having 1 post)

```
aggregate([
    {'$group':{'_id':'$user', 'count':{'$sum': 1}}},
    {'$match':{'count': {'$eq': 1}}},
    {'$group': {'_id': 'having 1 post', 'count':{'$sum': 1}}}
])
```

```
[{'_id': 'having 1 post', 'count': 66}]
```

User of the year

```
aggregate([
    {'$group':{
        '_id':{'year': { '$year': '$timestamp' }, 'user': '$user'},
        'count':{'$sum': 1}
    }},
    { '$sort': { '_id.year': 1, 'count': -1 }},
    { '$group':{
        '_id':'$_id.year',
        'first': {'$first': {'user': '$_id.user','count': '$count'}}
    }},
    { '$sort': { '_id': 1}}
])
```

```
[{'_id': 2008, 'first': {'count': 74, 'user': 'Sturla'}},
 {'_id': 2009, 'first': {'count': 617, 'user': 'arconaut'}},
 {'_id': 2010, 'first': {'count': 1306, 'user': 'arconaut'}},
 {'_id': 2011, 'first': {'count': 37627, 'user': 'teologovsan'}},
 {'_id': 2012, 'first': {'count': 8628, 'user': 'georg troyan'}},
 {'_id': 2013, 'first': {'count': 60846, 'user': 'georg troyan'}},
 {'_id': 2014, 'first': {'count': 170587, 'user': 'georg troyan'}},
 {'_id': 2015, 'first': {'count': 7742, 'user': 'georg troyan'}},
 {'_id': 2016, 'first': {'count': 15889, 'user': 'InnerIn'}},
 {'_id': 2017, 'first': {'count': 7015, 'user': 'Vulpes-Vulpeos'}}]
```

Number of data entered by a year

```
aggregate([
    {'$group':{
        '_id': { '$year': '$timestamp' }, 'count':{'$sum': 1}
    }},
    {'$sort': { '_id': 1}}
])
```

```
[{'_id': 2008, 'count': 74},
 {'_id': 2009, 'count': 656},
 {'_id': 2010, 'count': 1410},
 {'_id': 2011, 'count': 58483},
 {'_id': 2012, 'count': 16947},
 {'_id': 2013, 'count': 71637},
 {'_id': 2014, 'count': 175072},
 {'_id': 2015, 'count': 18177},
 {'_id': 2016, 'count': 38620},
 {'_id': 2017, 'count': 7444}]
```

Number of changeset with only one changed element

A changeset consists of a group of changes made by a single user over a short period of time.

```
aggregate([
    {'$group': {'_id': '$changeset', 'count': {'$sum': 1}}},
    {'$match': {'count': 1}},
    {'$group': {'_id': 'only one change', 'count': {'$sum': 1}}}
])
```

```
[{'_id': 'only one change', 'count': 989}]
```

Largest changeset in this region

```
aggregate([
    {'$group': {'_id': '$changeset', 'elemens': {'$sum': 1}}},
    {'$sort': {'elemens': -1}},
    {'$limit': 1}
])
```

```
[{'_id': 9366778, 'elemens': 4914}]
```

Number of new elements in the largest changeset

```
aggregate([
    {'$match': {'changeset': 9366778, 'version': 1}},
    {'$group': {'_id': 'new elements', 'count': {'$sum': 1}}}
])
```

```
[{'_id': 'new elements', 'count': 15}]
```

Number of changes by type in the largest changeset

```
aggregate([
    {'$match': {'changeset': 9366778}},
    {'$group': {'_id': '$type', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}}
])
```

```
[{'_id': 'node', 'count': 4910},
 {'_id': 'way', 'count': 3},
 {'_id': 'relation', 'count': 1}]
```

About the tags

Top 3 appearing amenities

```
aggregate([
    {'$match': {'tags.amenity' : { '$exists': True } }},
    {'$group': {'_id': '$tags.amenity', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}},
    {'$limit': 3}
])
```

```
[{'_id': 'parking', 'count': 149},
 {'_id': 'hospital', 'count': 65},
 {'_id': 'school', 'count': 59}]
```

Top 3 shops

```
aggregate([
    {'$match': {'tags.shop' : { '$exists': True } }},
    {'$group': {'_id': '$tags.shop', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}},
    {'$limit': 3}
])
```

```
[{'_id': 'convenience', 'count': 79},
 {'_id': 'supermarket', 'count': 30},
 {'_id': 'kiosk', 'count': 23}]
```

Top 5 appearing name's of street

```
aggregate([
    {'$match': {'tags.addr:street' : { '$exists': True } }},
    {'$group': {'_id': '$tags.addr:street', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}},
    {'$limit': 5}
])
```

```
[{'_id': 'Проскурівська вулиця', 'count': 138},
 {'_id': 'Миру проспект', 'count': 84},
 {'_id': 'Подільська вулиця', 'count': 81},
 {'_id': 'Грушевського вулиця', 'count': 81},
 {'_id': 'Соборна вулиця', 'count': 76}]
```

Top 3 type's of leisure

```
aggregate([
    {'$match': {'tags.leisure' : { '$exists': True } }},
    {'$group': {'_id': '$tags.leisure', 'count': {'$sum': 1}}},
    {'$sort': {'count': -1}},
    {'$limit': 3}
])
```

```
[{'_id': 'pitch', 'count': 71},
 {'_id': 'garden', 'count': 62},
 {'_id': 'stadium', 'count': 16}]
```

About the map

If I want to visit the central park of Khmelnitsky where can I charge my car at this time?

```
aggregate([
    {'$geoNear': {
        'near': { 'type': 'Point', 'coordinates': [ 26.97257, 49.43154 ] },
        'distanceField': 'distance',
        'query': {'tags.amenity': 'charging_station'},
        'includeLocs': "location",
        'limit': 3,
        'spherical': True
    }},
    {'$project':{'_id': 0, 'distance':1, 'location.coordinates': 1}}
])
```

```
[{'distance': 814.3856324451427,
  'location': {'coordinates': [26.9801727, 49.4261483]}},
 {'distance': 847.8606041482592,
  'location': {'coordinates': [26.9812475, 49.4264254]}},
 {'distance': 1262.6767588361008,
  'location': {'coordinates': [26.981175, 49.4216741]}}]
```

The best choice is number two, and I can find the way from charging station to the park.

Conclusion

At first glance, the data are the result of the work of communities contain many errors and for checking their need much time. But really, everything was easier.

The attributes of elements had no problems. Only geolocation data required the particular form for record data to JSON-file.

Then I investigated 442 kinds of tags. At first, I rejected tags that can be automatically converted to numbers. Then I dismissed the tags that have only one value and tags of objects in foreign languages that I do not know.

Check the tags using external sources also brought success. To test the data for loading into MongoDB, I used the SQLite database from the taginfo.openstreetmap.org site and rejected 33 tags that thoroughly described in the wiki and did not contain unnecessary information.

As a result, 81 tags left to handle manually, of which only 33% contained more than ten values.

After loading data into MongoDB and viewing statistics, it appears that information about the region in which I live is not complete. Very few objects are attractive to consumers. I think the reason for this is the poor activity of communities. Only 3 of the 263 users contributed almost 80% of the information. Peak data entry was in 2014 and now decreases.

But the primary goal achieved - information from an open source that made by communities can prepay and use for various tasks. Particularly, in this case, I interested to make requests about location data and calculating distances on the map.

What can be done to improve the data set?

If we consider the specific data set of the map, we can use the Notes API to load comments to incomplete data (for example, for the houses that are marked on the map and not have a number ask the number) and request the community to assist in completing the required data.

Advantages:

- the corrected data immediately sends on the site, and this information will be available for the next download
- Notes API allow tracking how the community is correcting data and seeing which items are closed and are not.

Disadvantages:

- the community can not immediately edit all notes, it may take a long time, and all data will not ready to the desired time.
- the fixed data must again pass through the procedure of preparation, review, and download.

If I will dream about improving the data on the site, it is possible to develop a game similar to Pokemon GO app that will combine data from maps and stream of the camera. The goal of the game, instead of collecting Pokemon hunt the white spots on the map and bring information about them to the site using a GPS. After the introduction of related information about objects, gamers gain points and become the owners of the facility in the virtual space. They can sell them or exchange on other artifacts. Players can unite in groups. They may fight among themselves. They may hunt the new facilities on the map, may fix inaccuracies of other users or add new relevant related information of old objects.

There may be not active users who can watch the results of the game. For example pointing a smartphone at a building, to see the information about him. The picture is something like the Terminator seen: house number, owner, information about companies inside the building, and other.

Advantages:

- invitation new members of the community
- data will input directly from the local place using GPS

Disadvantages:

- collected data may contain many inaccuracies because the time affect the outcome of the game
- Received data belong to certain classes of objects that will be limited the size of smartphone's screen. Information about buildings will be entered easy but will complex task collect data for roads, forests, lakes and other big objects.
- will be hard to maintain the activity of players when white spots on the maps will tend to end

References

- OpenStreetMap
- OpenStreetMap Documentation
- OSM XML Documentation
- Taginfo download page
- ITU-T E.164 - international format for numbers phone
- Notes API