Project Report:
Data Wrangling with MongoDB

Zai Feng Wang

6/18/2015

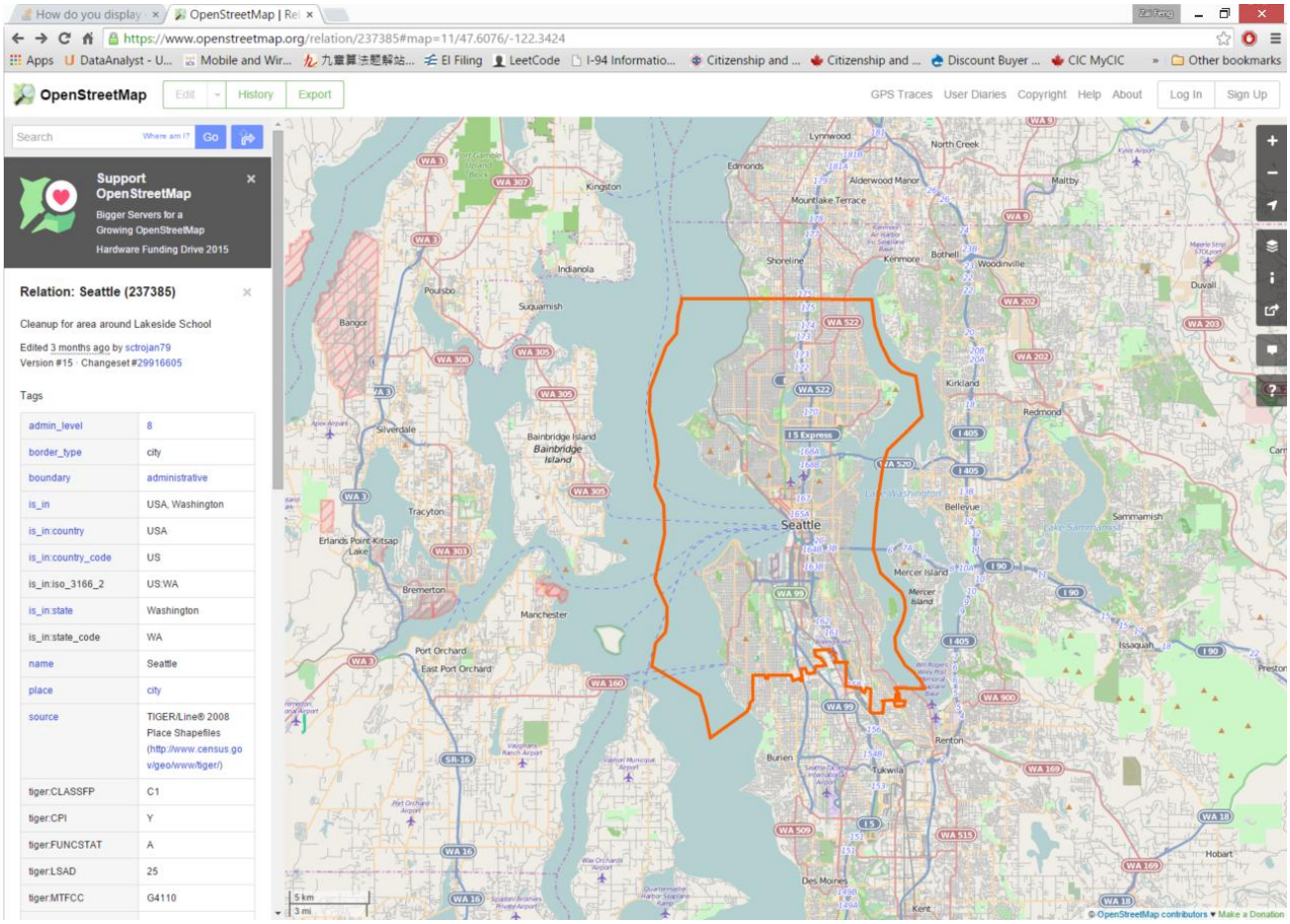
Table of Contents

1. Description 1
2. Problems Encountered in the Map 1
   a. Abbreviated Street Names 1
   b. Inconsistent zip code 2
   c. Unexpected zip code 2
3. Data Overview 2
   # File size: 2
   # Number of tags in the dataset: 2
   # Number of each types of k value in tag ==’tag’: 3
   # Number of nodes: 4
   # Number of ways: 5
   # Number of unique users: 5
   # Top 1 contributing user: 5
   # Number of users appearing only once (having 1 post) 5
   # Top two zipcodes having the highest count 5
   # Top two cities having the highest count 6
4. Additional Ideas on user contribution 6
5. Additional data exploration using MongoDB queries 7
   # Top 10 appearing amenities 7
   # Biggest religion 7
   # Most popular cuisines 7
6. Conclusion 7
7. References 8

1. Description

Here is the link to the map of Seattle WA which I selected for this project.

https://mapzen.com/data/metro-extracts



The reason I selected this area is pretty simple - I live in the metropolitan area of Great Seattle.

2. Problems Encountered in the Map

Once the seattle_washington.osm.json file is imported into MongoDB, I did some basic query and realized some issues embedded in the map file as bellows:

- Abbreviated street names
- Inconsistent zip code
- Unexpected zip code

a. Abbreviated Street Names

With some basic queries of the MongoDB database, I found that many street names end with abbreviation which does not follow consistent rule, resulting in unstructured data which could cause undesired query result. For example, St and St. are used to represent Street. I updated all street names as per below mapping rule:

```
mapping = {
                "St": "Street",
                "St.": "Street",
                "Rd.": "Road",
                 "Ave": "Avenue"
            }
```

b. Inconsistent zip code

The postcodes format in Seattle WA is 5 digits starting with 98 or 99, however the above search shows many incorrect formats such as u'WA 98102', u'98501-5801', u'Elma, 98541', u'Olympia, WA 98502'. I have stripped all leading and trailing characters and updated all records as per standard zip code for Seattle WA. The key code is as below:

```
# find postcode with leading characters like "WA 98053", "Elma, 98541", "Olympia, WA 98502"
leading_chars_postcode = re.compile(r'\s\d{5}\Z')
# find postcode with trailing characters like '98501-5801'
trailing_chars_postcode = re.compile(r'\A\d{5}-\d{4}\Z')
# clean up postcode by removing leading and trailing characters
def update_postcode(postcode):
    m_leading = leading_chars_postcode.search(postcode)
    m_trailing = trailing_chars_postcode.match(postcode)
    if m_leading:
        postcode = postcode[-5:]
    elif m_trailing:
        postcode = postcode[:5]
    return postcode
```

Please see detail in attached AuditAndClean_updated.py

c. Unexpected zip code

The last thing is this map also includes many cities from British Columbia, Canada which has a different format of postcode like 'V8W 1H8'. They are not wrong postal code but just due to the fact that this map is actually for Metropolitan Area and not expected.

3. Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

 # File size:

Seattle_washington.osm: 1,402,530KB

Seattle_washington.osm.json: 1,588,179KB

 # Number of tags in the dataset:

```
#find the number of tags in this dataset
import xml.etree.ElementTree as ET
import pprint
import os
def count_tags(filename):
        tags = {}
        context = ET.iterparse(filename,events=("start","end"))
        context = iter(context)
        event, root = context.next()
        for event, elem in context:
            if event == "end" and elem.tag in tags:
                tags[elem.tag] += 1
            elif event == "end" and elem.tag not in tags:
                tags[elem.tag] = 1
            root.clear()
        return tags
file_name = 'seattle_washington.osm'
file_path = 'D:\wzf\DataAnalyst\DataMango\FinalProject'
filename = os.path.join(file_path,file_name)
tags = count_tags(filename)
pprint.pprint(tags)
```

```
{'bounds': 1,
 'member': 35858,
 'nd': 7096564,
 'node': 6435097,
 'osm': 1,
 'relation': 5120,
 'tag': 4199754,
 'way': 605700}
```

 # Number of each types of k value in tag ==’tag’:

```
#Number of types of k value in tag == 'tag'
import xml.etree.ElementTree as ET
import pprint
import re
import os

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":
        # YOUR CODE HERE
        value_k = element.get('k')
        if lower.match(value_k):
            keys['lower'] += 1
        elif lower_colon.match(value_k):
            keys['lower_colon'] += 1
        elif problemchars.search(value_k):
            keys['problemchars'] += 1
        else:
            keys["other"] += 1
    return keys

def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    context = ET.iterparse(filename,events=("start","end"))
    context = iter(context)
    event, root = context.next()
    for event, elem in context:
        if event == "end":
            keys = key_type(elem, keys)
        root.clear()
    return keys

file_name = 'seattle_washington.osm'
file_path = 'D:\wzf\DataAnalyst\DataMango\FinalProject'
filename = os.path.join(file_path,file_name)
keys = process_map(filename)
pprint.pprint(keys)
```

```
{'lower': 1890275,
 'lower_colon': 2237169,
 'problemchars': 7,
 'other': 72303,}
```

 # Number of nodes:

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.test
result = db.seattle.find({'type':'node'}).count()
print(result)
```

```
6435065
```

 # Number of ways:

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.test
result = db.seattle.find({'type':'way'}).count()
print(result)
```

```
605578
```

 # Number of unique users:

```
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.test
result = db.seattle.aggregate([{'$group':{'_id':'$created.user'}}])
count = 0
for i in result:
   count +=1
print count
```

```
2580
```

 # Top 1 contributing user:

```
db.seattle.aggregate([{'$group':{'_id':'$created.user','count':{'$sum':1}}},\
                      {'$sort':{'count':-1}},\
                      {'$limit':1}])
```

```
{u'count': 1125969, u'_id': u'Glassman'}
```

 # Number of users appearing only once (having 1 post)

```
db.seattle.aggregate([{'$group':{'_id':'$created.user','count':{'$sum':1}}},\
                      {'$group':{'_id':'$count','num_users':{'$sum':1}}},\
                      {'$sort':{'_id':1}},\
                      {'$limit':1}])
```

```
{u'num_users': 542, u'_id': 1}
```

 # Top 2 zipcodes having the highest count

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

The top two zipcodes having the highest count are as below:

{u'_id': u'98034', u'count': 22968}

{u'_id': u'98033', u'count': 19382}

They are actually located in Kirkland WA which is not far away from where I'm living.

 # Top two cities having the highest count

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

The top two cities having the highest count are as below:

{u'_id': u'Seattle', u'count': 202708}

{u'_id': u'Kirkland', u'count': 42317}

4. Additional Ideas on user contribution

```
#top 10 users that have the highest contribution
from pymongo import MongoClient
import pprint
client = MongoClient()
db = client.test
result = db.seattle.aggregate([{'$group':{'_id':'$created.user','count':{'$sum':1}}},\
                               {'$sort':{'count':-1}},\
                               {'$out':'user_count'}])
re = db.user_count.find().limit(10)
for i in re:
   print i
re2=db.user_count.aggregate([{'$group':{'_id':'null',\
                                        'total':{'$sum':'$count'},\
                                        'average_contribution_per_user':{'$avg':'$count'}}}])
for j in re2:
   print j
```

Top user contribution percentage ("Glassman") – 16%

Combined top 2 users' contribution ("Glassman" and "SeattleImport") – 26.7%

Combined top 10 users contribution - 63.2%

Average_contribution_per_user – 2729/3.9%

The contributions of users seems much skewed, this could possibly due to the fact that some users have to manually edit the map while others do this automatically. There are two ways to improve this – either provide gamification elements such as rewards, gifts, badges, or create more efficient bots. We may write some programs to automatically increase the points of the users when they make every input. Once the points reach to some point, we can reward them with gift cards, badge, etc. the

issue is the reward won't last long and is only a temporary solution. To find a long term solution may take time to discover and won't be covered in this project.

5. Additional data exploration using MongoDB queries

 # Top 10 appearing amenities

```
db.seattle.aggregate([{'$match':{'amenity':{'$exists':1}}},\
                      {'$group':{'_id':'$amenity','count':{'$sum':1}}},\
                      {'$sort':{'count':-1}},\
                      {'$limit':10}])
```

```
{u'count': 6447, u'_id': u'parking'}
{u'count': 2997, u'_id': u'bicycle_parking'}
{u'count': 2884, u'_id': u'school'}
{u'count': 2227, u'_id': u'restaurant'}
{u'count': 1455, u'_id': u'bench'}
{u'count': 1392, u'_id': u'place_of_worship'}
{u'count': 967, u'_id': u'fast_food'}
{u'count': 873, u'_id': u'cafe'}
{u'count': 866, u'_id': u'fuel'}
{u'count': 655, u'_id': u'toilets'}
```

 # Biggest religion

```
db.seattle.aggregate([{'$match':{'amenity':{'$exists':1},'amenity':'place_of_worship'}},\
                      {'$group':{'_id':'$religion','count':{'$sum':1}}},\
                      {'$sort':{'count':-1}},\
                      {'$limit':1}])
```

```
{u'count': 1291, u'_id': u'christian'}
```

 # Most popular cuisines

```
db.seattle.aggregate([{'$match':{'amenity':{'$exists':1},'amenity':'restaurant'}},\
                      {'$group':{'_id':'$cuisine','count':{'$sum':1}}},\
                      {'$sort':{'count':-1}},\
                      {'$limit':2}])
```

```
{u'count': 777, u'_id': None}
{u'count': 178, u'_id': u'american'}
```

6. Conclusion

After I reviewed this Seattle WA map data I found this actually include more unexpected British Columbia cities in Canada. In addition, some data are still missing like the #1 popular cuisine, but I believe the original data has been cleaned enough to make a clear data base for this purpose of project. To improve these, some constraints need to be applied to not allow postcode with Canadian format into the dataset. We may write some script to automate this judgement. For the latter case, those

restaurant amenities should be omitted if they have 'None' for 'cuisine', this easily lead to the result that American cuisine is the most popular one in Seattle area. I have audit and cleaned street names, reshaped the dataset and standardized zip codes. I've found top ten amenities and most popular cuisines in this city. It was great to work on map data for the first time.

7. References

http://docs.python-requests.org/en/latest/

https://docs.python.org/2/library/csv.html

http://pymotw.com/2/json/

http://www.w3schools.com/xml

http://www.crummy.com/software/BeautifulSoup/bs4/doc/

https://docs.python.org/2/library/re.html

https://docs.python.org/2/howto/regex.html#regex-howto

http://api.mongodb.org/python/current/tutorial.html

http://docs.mongodb.org/manual/reference/operator/aggregation/group/

http://docs.mongodb.org/manual/reference/program/mongoimport/

https://docs.python.org/2/library/functions.html#enumerate

http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-file-generated-via-xml-etree-elementtree-in-python

https://mail.python.org/pipermail/xml-sig/2005-January/010838.html

http://eli.thegreenplace.net/2012/03/15/processing-xml-in-python-with-elementtree