OpenStreetMap Project: Data Wrangling with MongoDB

Map Areea: Athens, Greece
Downloaded: https://s3.amazonaws.com/metro-extracts.mapzen.com/athens_greece.osm.bz2 on August 19, 2015 at 10:24am

1. Problems Encountered in the Map

After initially downloading the full sample size of the Athens metropolitan area and running it against the CleanProcess.py file, I noticed three main problems with the data, which I will discuss in the following order:

- Abbreviated street names (`“Kifissias Ave”`)
- Abbreviated city names (`“Athen”`)
- Incorrect postal codes (`“1344”`) (needs to be a 5 digit code)
- Inconsistent phone numbers (`“()210-3213488, -210 321 8733”`)

To audit the osm file, first we need to know the overview of the data. To get an overview of the data, we count the tag content of the data.

```
{'bounds': 1,
 'member': 54998,
 'nd': 1827038,
 'node': 1548656,
 'osm': 1,
 'relation': 4361,
 'tag': 496675,
 'way': 166453}
```

Then we need to exclude and find all be problematic tag inputs. After running the equivalent code we get. The reason why we need to exclude them is because MongoDB won’t be able to recognize the JSON array with these characters included.

```
{'lower': 414455, 'lower_colon': 79100, 'other': 3081, 'problemchars': 39}
```

OpenStreetMap is powerful, but relies heavily on human input. As a result, due to the human input, there is higher chance for error. An extra drawback to the analysis is that Python 2.7 does not recognize certain the Greek characters and as a result the UTF-8 encoding translates them as a sequence similar to the following ”`u'\u0386\u03b3\u03b9\u03bf\u03b9”.` The way I dealt around this issue is by focusing exclusively on the Latin characters and ignore the Greek ones.

Abbreviated street names

Similar to the audit.py file from the course materials, certain inconsistent abbreviations are used for street types and correct them to the appropriate full name. Such examples are the following:

```
'Ave': set(['Kifissias Avenue', 'Poseidonos Avenue']),
 'St': set(['Kifissou Street', 'Trapezountos Street']),
 'St.': set(['22 Ag. Asomaton & 12 Dipilou Street']),
 'Str': set(['Kazantzaki Street']),
 'Str.': set(['Filadelfeos Street']),

 u'road': set([2nd km Kapandriti \ Kalamos Road'])}
```

Code

```
def clean_street_name(street_name, mapping,street_types):
      m = street_type_re.search(street_name)
      if m:
            street_type = m.group()
            if street_type in mapping:
                  street_name=re.sub(street_type_re,mapping[street_type,stre
                  et_name)
                  street_types[street_type].add(street_name)
            #if street_type not in expected:
            #street_types[street_type].add(street_name)
      return street_name
```

Abbreviated/Incorrect city names

Second type of corrections I did was on the city names. Several city names had incorrect structure starting with "`u'\u0386\u03b3\u03b9\u03bf\u03b9`" and obviously they are excluded from the sample. Some of the corrected abbreviated examples are the following:

```
{'Athen': set(['Athens']), 'Korvp;i': set(['Koropi'])}
```

Code

```
def clean_city(city_name, city_map, city_types):
      m = street_type_re.search(city_name)
      if m:
            city_type=m.group()
            if city_type in city_map:
                  city_name= re.sub(street_type_re,city_map[city_type] , city_name)
                  city_types[city_type].add(city_name)
            #if city_type not in city_exp:
                  #city_types[city_type].add(city_name)
      return city_name
```

Incorrect postal codes

I screened all postal codes for having only numbers and no other characters and for being 5 digits as they all are in Greece. I discovered one case where there is an incorrect postal code and I excluded it from the JSON file.

```
{'1344': set(['1344'])}
```

Code
```
    elif key == "postcode":
       value = re.sub("[^0-9]", "", value)
       if len(value) != 5:
              zipcode_types[value].add(value)
```

Inconsistent phone numbers

Finally I explored the phone numbers and tried to identify the cases where they were not written in the proper format or were purely incorrect. I identified the following cases of problematic phone numbers:
```
{'()210-3213488': set(['(+30)210-3213488']),
 '+030 699 708 0766': set(['+030 699 708 0766']),
 '+31 210 2531867': set(['+31 210 2531867']),
 '-210 321 8733': set(['+30-210 321 8733']),
 '-210-3805611': set(['+30-210-3805611'])}
```
Code
```
def clean_phone(key, value, phone_types):
       #Athens phone numbers have a 3 digit area code(210) following for a 7 digit
number
       #if only 7 digits are supplied, I assume it's missing the Athens area code
       list_of_phone_num = []
       phone_numbers = value.split(";")
       for number in phone_numbers:
              number = number.replace("+30","").lstrip().rstrip()
              m=re.match(r"[^0-9]",number)
              if m:
                     phone_types[number].add(value)
              numbers_only = re.sub("[^0-9]", "", value)
              if len(numbers_only) == 7 :
                     number = "210" +  number
              try:
                     phone_num = pn.parse(value,"GR")
                     if pn.is_valid_number(phone_num):
                     cleaned=pn.format_number(phone_num,pn.PhoneNumberFormat.INTERNATI
              ONAL)
                     list_of_phone_num.append(cleaned)                    except:
                     pass
       return list_of_phone_num
```

2. Data Overview

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

File sizes

```
athens_greece.osm ......... 326 MB
sample_athens_greece.osm ... 33 MB
athens_greece.osm.json .... 480 MB
```

Some of the results from the Pymongo queries I built.

```
db.map.find().count()
```

Number of documents

```
1715109
============================================================
db.map.find({"type":"node"}).count()
```

Number of nodes

```
1548656
============================================================
db.map.find({"type":"way"}).count()
```

Number of ways

```
166453
============================================================
def uniqueusers():
pipeline = [
      {"$group": {"_id": "$created.user"}},
      {"$group": {"_id":1, "count":{"$sum":1}}}
]
result =db.map.aggregate(pipeline)
```

Number of unique users

```
 [{u'_id': 1, u'count': 1153}]
============================================================
def userContributions():
pipeline = [
      {'$match': {"created.user":{'$exists':1}}},
      {"$group": { "_id" : "$created.user",  "count" : {"$sum" : 1 }}},
      {"$sort": {"count" : -1}},
      {"$limit" : 10}
]
result = db.map.aggregate(pipeline)
```

Top 10 users by contributions

```
[{u'_id': u'makmar', u'count': 651699},
 {u'_id': u'greecemapper', u'count': 342158},
 {u'_id': u'mtraveller', u'count': 108524},
 {u'_id': u'NikosSkouteris', u'count': 80249},
 {u'_id': u'aitolos', u'count': 68685},
 {u'_id': u'Amaroussi', u'count': 63594},
 {u'_id': u'Chris Makridis', u'count': 53069},
 {u'_id': u'Kanenas', u'count': 37199},
 {u'_id': u'athinaios', u'count': 34841},
 {u'_id': u'AiNikolas', u'count': 32533}]
============================================================
def singlepostusers():
pipeline = [
      {"$group":{"_id":"$created.user", "count":{"$sum":1}}},
      {"$group":{"_id":"$count", "num_users":{"$sum":1}}},
      {"$sort":{"_id":1}}, {"$limit":1}
]
result =db.map.aggregate(pipeline)
```

Number of users having only 1 post

```
[{u'_id': 1, u'num_users': 218}]
============================================================
def countAddresses():
```

```
return db.map.count( {"address" : {"$exists" : 1}}  )
Total number of addresses
13992
============================================================
def count_incomplete_addresses(field):
pipeline = [
      {"$match":  {"address" : {"$exists" : 1}, "address." + field :
{"$exists" : 0}}},
      {"$group": {"_id": "Addresses", "count" : {"$sum" : 1}}},
]
result = db.map.aggregate(pipeline)
pprint.pprint(list(count_incomplete_addresses("street")))
Addresses with no Street
[{u'_id': u'Addresses', u'count': 1954}]
============================================================
pprint.pprint(list(count_incomplete_addresses("postcode")))
Addresses with no zipcode
[{u'_id': u'Addresses', u'count': 5629}]
============================================================
pprint.pprint(list(count_incomplete_addresses("housenumber")))
Addresses with no House number
[{u'_id': u'Addresses', u'count': 1811}]
============================================================
def addressGroupByCitySuburb():
pipeline = [
      {"$match":  {"address" : {"$exists" : 1}}},
      {"$group": {"_id": {"city": "$address.city", "suburb" :
"$address.suburb" }, "count" : {"$sum" : 1}}},
      {"$sort": {"count" : -1}},
      {"$limit" : 5}
]
result = db.map.aggregate(pipeline)
Top 10 addresses by City
[{u'_id': {u'city': u'\u0391\u03b8\u03ae\u03bd\u03b1'}, u'count': 7388},
{u'_id': {}, u'count': 3195},
{u'_id': {u'city': u'\u0391\u03c7\u03b1\u03c1\u03bd\u03ad\u03c2'}, u'count':
1752},
{u'_id': {u'city': u'\u03a0\u03b5\u03b9\u03c1\u03b1\u03b9\u03ac\u03c2'},
u'count': 666},
{u'_id': {u'city': u'\u039a\u03b1\u03bb\u03bb\u03b9\u03b8\u03ad\u03b1'},
u'count': 162}]
```

3. Additional Ideas

So we can deduce some interesting statistics from the above generated data.

- The Top 10 contribution users generated 1,472,551 entries, which represents 86% of the total
- Top User contribution was 38% of the total
- The users that have submitted only one post is 19% of the total users

- The number of addresses with not street name (or unrecognizable street name only in Greek letters) is 14% of the total
- The number of addresses without a zipcode is 40% of the total
- The number of addresses without a number is 13% of total
- The top street appears 7,388 times in the dataset

The OpenStreetMap project could be improved by 2 online "yellow pages" websites which operate in Athens and have information about all type of "amenities" that operate in the city. These websites are vrisko.gr and xo.gr. However, as this information is proprietary to these websites, there may be a conflict of interest and they would need some sort of publicity or purchase of the data from the OpenStreetMap project. Using such datasets would also be useful for cross-referencing the locations of all the "amenities" in the OpenStreetMap data.

Another idea for improving the data set is using the Greek taxi-hailing App called Taxibeat. There are thousands of taxis taking pick-up calls in an hourly basis in Athens and the data provided from the users of the App would certainly be accurate to indicate the location of arrival and departure. Incorporating such data into the OpenStreetMap project would be another great solution in improving the dataset.

4. Conclusion

After this review of the data it's obvious that the Athens, Greece area is incomplete. However the biggest impediment to my analysis has been the inability to encode correctly the Greek letters which represent the majority of the streets. Based on comments that I read this is a limitation of the Python 2.7 version, which may have been improved in the 3.4. Developing in the future a more robust data processor will be a next step in cleaning and processing a better number of data from the OpenStreetMap database.