DANDP3

Wrangling Data from OSM with MongoDB

Data Analyst Nanodegree

Anna Signor

Process Overview

I followed the folowing steps:

- download XML from Open Street Map, using mapzen
- with Python scripts check for problems in the data
- adjust code accordingly
- parse and shape data into one JSON file
- import data into MongoDB
- through datbase queries, check for remaining problems
- repeat last 4 steps until data is acceptable

Area

The area chosen is São Paulo, Brazil, where I was born and raised. The XML dataset was downloaded from MapZen. My intention was to download and explore the São Paulo Metropolitan Area, this map relation. As is detailed in a further section, I found that this is a rather ambiguous term, there being at least two different concepts that translate into that from Portuguese. The conclusion is the area analysed is called *Complexo Metropolitano Expandido* or Expanded Metropolitan Complex of São Paulo, a considerably larger area than what I initially thought I had downloaded.

Examples of Problems Uncovered Before Querying DB

1. Street Names

Using an **audit** function two distinct classes of problems were uncovered with the street names:

street types uppper/lower case or abbreviations inconsistencies, mispellings: E. g. the words "Rua", "R.", "RUA", "Rue" and "rua" all occurred

street types mising : E. g. "Alfonso Bovero" where "Avenida Alfonso Bovero" should be

Both issues were addressed by the function **improve_names_BR** below. Information elucidaded by **audit** was fed back into the code.

In [ ]:

```
mapping = {'avenida':'Avenida', #mapping to fix case mispellings, case, abbreviations
           u'Al.': 'Alameda',
           'Rue': 'Rua',
           u'Av.': 'Avenida',
           u'Av': 'Avenida',
           'RUa': 'Rua',
           'R': 'Rua',
           'Acost.': 'Acostamento',
           'RUA': 'Rua',
           'rua' : 'Rua',
           'R.' : 'Rua',
           'AC': 'Acesso',
           'estrada' : 'Estrada',
           'travessa' : 'Travessa'
           }
good_types = set(['Acostamento', #Set of acceptable street types
                  u'Pra\xe7a',
                  'Alameda',
                  'Viela',
                  'Estrada',
                  'Rua',
                  'Acesso',
                  'Parque',
                  'Largo',
                  'Via',
                  'Marginal',
                  'Rodovia',
                  'Corredor',
                  'Viaduto',
                  'Travessa',
                  'Pateo',
                  'Avenida',
                  'Passagem',
                  u'Complexo Vi\xe1rio'])
mapping2 = {u'1\xaa Travessa da Estrada do Morro Grande' : '', #mapping for case when street type is missin
g
            'Alfonso Bovero' : 'Avenida',                     #this was manually created by looking up all
the names
            u'N\xedvia Maria Dombi' : 'Travessa'              #the ones not showing here are "Rua" types
            }

good_tuple = tuple(good_types)

def improve_name_BR(name):
    """takes a street name from sao paulo and returns improved name"""
    words = name.split()
    if name.startswith(good_tuple):
        return name
        ### if name is okay, return name (do nothing)
    elif words[0] in mapping:
        words[0] = mapping[words[0]]
        return ' '.join(words)
        ### if type is mispelled or miscased, update 1st word of name and return joined string
    elif name in mapping2:
        return mapping2[name] + ' ' + name
        ### if name is one of the odd cases but not needing word 'Rua', use mapping2 to fix
    else:
        return 'Rua' + ' ' + name
        ### the cases left are the ones where the word 'Rua' was left out
        ### this choice was made because "Rua" is the most commonly occurring type
```

2. Data Structure

The data structure was interesting. It may be adequate for OSM, but it is certainly not how I would like for it to figure in the MongoDB collection. Some information is represented directly as attributes of a main XML data primitive element, while others, as attributes of child elements tagged "tag". A choice was made by OSM to have attributes called "k" and "v", the values of which represent keys and values, rather than using an XML of the type`<key> value <key/>`. Furthermore, there was use of a colon hierarquical structure in some of the "k"s. Here is an example node:

```
    <node changeset="38648623" id="4128352041" lat="-23.5591903" lon="-46.6587486" timestamp="2016-04-17
    T17:59:02Z" uid="2030995" user="Bonix-Mapper" version="1">
            <tag k="name" v="Banca Paulista V" />
            <tag k="shop" v="books" />
            <tag k="phone" v="+55 11 3288-8241" />
```

```
            <tag k="addr:street" v="Avenida Paulista" />
        </node>
```

I would like to represent the data in JSON in a different shape, so this was something that had to be considered while parsing. To put it simply: although a very simple and straight-forward mapping to translate XML into JSON is always possible, this is not what I used because the data was not in a desireable shape. This is what the function **shape_element** mostly does.

The shape chosen to represent the data is the following:

```
    {
    "id": value,
    "data_prim": node_way_or_relation,
    "visible":true_or_false,
    "created": {
              "version":value,
              "changeset":value,
              "timestamp":value,
              "user":value,
              "uid":value_for_user
            },
    "pos": [LAT, LON],
    "address": {
              "housenumber": value,
              "postcode": value,
              "street": value
                ...
            },
    "amenity": value,
    "cuisine": value,
    "name": value,
    "phone": value
    "any_other_attrib_1": value
    ...
    "any_other_attrib_n": value
    "A" : {
            "B" : {
                    "C" : value
                    }
            }
    }
```

where "A", "B" and "C" represent the `<tag k="A:B:C" v=value />` value situations that are not the address, which is a special case.

Most of this was hard-coded into **shape_element**, and to deal with the colons I used a helper function called **smarter_nestify**. (Which actually allows for processing a key containing an arbitrary number of colons into nested dictionaries, using regression.)

In [1]:

```
def smarter_nestify(l, record):
    """Takes a list [a1, a2, a3, ... , an, value] and a pre-existing dictionary structure returns a nested d
ictionary object
    {a1 : {a2 : {a3 : ... {an : value} ...}}}, respecting the pre-existing dictionary records, that is, for
each recursion
    step if a dictionary ai already exists it will add a key ai+1 to it rather than creating a new dictionar
y ai."""
    if len(l) == 2: #if list is down to two elements [a, val], return {a : val}
        key = l[0]
        value = l[1]
        return {key: value}
    else:
        key = l[0]
        record[key] = smarter_nestify(l[1:], record.get(key, {}))
        return record
    # function pops the first element of the list, makes a dictionary {k : v} where k is the popped element
and v is what is
    # returned when calling itself on popped list and empty dictionary or existing one, depending on record
    """adapted from:
    http://stackoverflow.com/questions/37014500/how-to-use-recursion-to-nest-dictionaries-while-integrating-
with-existing-record
    """
```

3. Repeated Attribute Keys

This was an interesting issue that stems from not discarding the tags containing colons. The problem is tag 'k' attributes that have different functions being called the exact same in the XML data. For example, we had:

```
<tag 'k'='lanes:psv:forward' 'v'='1'>
```

and

```
<tag 'k'='lanes' 'v'='2'>
```

in the same node. In the OSM XML schema, the first data has to do with lanes that have special permission (like a taxi, carpool or a bus lane) while the second is simply the number of lanes in any road. This was causing a variable type error. It was resolved by repeatedly trapping the error and using the information back into the code manually. Ref: https://discussions.udacity.com/t/keep-attr-atrr-atrr-formatted-data/166864/14

4. Variable types

The data was almost entirely represented in strings or unicode. A lot of the data will be more useful a different variable type. MongoDB supports all the types supported in Python, and since some of the calculations are numerical, it will be in my favor to convert types as the data is parsed. Conversions made:

- 'POS' into float
- 'version' into int

OBS: It is in my favor to treat postcode as a string since the zeroes to the left have significance, which an int type would ignore. Also, brazilian postcodes contain a non-numerical charachter "-", which would cause problems.

Examples of Problems Uncovered by DB Query

The cell below is preparation for DB query.

In [1]:

```
from pymongo import MongoClient
import pprint

client = MongoClient()
db = client
sp = db.my_osm.cme #shorthand since all the queries will be in same collection
```

1. Bad Key 'type' Creating Incorrect Parsing

At first, the desired data format had a 'type' key in the main JSON node, to designate a map node, way or relation (please note the word "node" here has two very different meanings). So I ran the query below to find out how many relations were in the dataset.

In [7]:

```
relations = sp.find({'type' : 'relation'})
```

The original output of this query was 4. It seems odd that a metropolis so big would have 4 relations. Investigating further, I found there were 261671 ways and 1900291 nodes.

In [8]:

```
4 + 261671 + 1900291 - 2168319 #this should return 0
```

Out[8]:

```
-6353
```

What this effectively means is that there is some kind of discrepancy, and it is not small.

In [26]:

```
types = sp.distinct('type')
pprint.pprint(types)
```

```
[u'enforcement',
 u'palm',
 u'triangulation',
 u'fixed_point',
 u'multipolygon',
 u'site',
 u'water',
 u'oil',
 u'route',
 u'gas',
 u'audio',
 u'boundary',
 u'restriction',
 u'waterway',
 u'associatedStreet',
 u'route_master',
 u'public_transport',
 u'bridge',
 u'tunnel']
```

The above output is totally unexpected, as we should see only "node", "relation" or "way". A little research was helpful in pinpointing the issue, mainly that relation data primitive were translating into nodes with incorrect 'type' assignment, because many of the relations themselves had a 'type' attribute or tag child. Full explanation on this reference. The forum post also contains the solution, which is to not use the word "type". I decided to call this key "data_prim" in short for "data primitive".

After fixing the Python code, making a new JSON file, clearig the old collection off the database and loading the new one, when we run the queries below the expected outputs are produced.

In [6]:

```
types = sp.distinct('data_prim')
pprint.pprint(types)
```

```
[u'node', u'way', u'relation']
```

In [7]:

```
cursor = sp.find({'data_prim' : 'way'})
a = len(list(cursor))
print a, 'ways'
cursor = sp.find({'data_prim' : 'node'})
b = len(list(cursor))
print b, 'nodes'
cursor = sp.find({'data_prim' : 'relation'})
c =  len(list(cursor))
print c, 'relations'
print 'discrepancy:', 2168319 - a - b - c
```

```
261695 ways
1900322 nodes
6302 relations
discrepancy: 0
```

2. Seamarks and unexpected postcodes

One of the strange things in my data was the occurrence of the tag "seamark". This is one of the features that make heavy use of the colons structure in the OSM XML schema, so it was in the back of my head. A simple query reveales how many of them there are in the data.

In [79]:

```
sp.find({'seamark' : {'$exists' : 1}}).count()
```

Out[79]:

```
143
```

The reason why this is strange is upon further investigation in the OSM Wiki, I found these are features that should occur in oceanic coasts. Querying the data base for examples I found some were lighhouses and buoyes. The problem is the metropolitan area I was supposed to be analysing contains NO sea coast. The query below shows how many and which cities figure in the dataset.

In [2]:

```
cities = sp.distinct('address.city')

cities.sort()
print cities
len(cities)
```

```
[u'Aldeia de Carapicu\xedba', u'Alum\xednio', u'Ara\xe7ariguama', u'Aruja', u'Aruj\xe1', u'Atibaia', u'Baru
eri', u'Bertioga', u'Bom Jesus dos Perd\xf5es', u'Cajamar', u'Campo Limpo Paulista', u'Carapicu\xedba', u'Co
tia', u'Cubat\xe3o', u'Diadema', u'Embu das Artes', u'Engenheiro Goulart', u'Ferraz de Vasconcelos', u'Franc
isco Morato', u'Franco da Rocha', u'Guaruja', u'Guaruj\xe1', u'Guarulhos', u'Ibi\xfana-SP', u'Igarat\xe1',
u'Indaiatuba', u'Itanha\xe9m', u'Itapacerica da Serra', u'Itapecerica da Serra', u'Itapevi', u'Itaquacetuba'
, u'Itaquaquecetuba', u'Itu', u'Itupeva', u'JUndia\xed', u'Jacare\xed', u'Jundiai', u'Jundia\xcc', u'Jundia
\xed', u'Jundi\xe1i', u'Mairipora', u'Mairipor\xe3', u'Maua', u'Mau\xe1', u'Mogi das Cruzes', u'Mogis das C
ruzes', u'Mooca', u'Osasco', u'Po\xe1', u'Ribeir\xe3o Pires', u'SA\xd5 PAULO', u'Salto', u'Santana de Parna
\xedba', u'Santo Andre', u'Santo Andr\xe9', u'Santos', u'Sao Bernardo do Campo', u'Sao Jose dos Campos', u'S
ao Paolo', u'Sao Paulo', u'Sao paulo', u'Suzano', u'S\xc3O PAULO-SP', u'S\xe0o Paulo', u'S\xe2o Paulo', u'S
\xe3o Bernardo do Campo', u'S\xe3o Bernardo do campo', u'S\xe3o Caetano do Sul', u'S\xe3o Jos\xe9 dos Campo
s', u'S\xe3o Pailo', u'S\xe3o Paulo', u'S\xe3o Paulo - SP', u'S\xe3o Paulo, SP', u'S\xe3o Paulo/SP', u'S\xe
3o Paulol', u'S\xe3o Roque', u'S\xe3o Vicente', u'S\xe3o jos\xe9 dos Campos', u'S\xe3o paulo', u'S\xe3o vic
ente', u'Tabo\xe3o da Serra', u'Vargem Grande Paulista', u'Varzea Paulista', u'V\xe1rzea Paulista', u'jd dos
cunhas', u'jundia\xed', u'santo Andr\xe9', u'sao paulo', u'sp', u's\xe3o Caetano do Sul', u's\xe3o Paulo',
u's\xe3o paulo']
```

Out[2]:

```
92
```

This seems to indicate the area is, in fact, what is called Expanded Metropolipan Complex of São Paulo or *Complexo Metropolitano Expandido* (in this terminology "São Paulo" is implied), also called Paulistan Macrometropolis or *Macrometrópole Paulista*. It is not, as I thought, *São Paulo Metropolitan Area*, or *Região Metropolitana de São Paulo (RMSP)*, also called Large São Paulo *Grande São Paulo*. The difference in area and concepts can be grasped by checking out the Wikipedia pages RMSP and CME for those who are interested. The important point to make is that this dataset is referring to the Expanded Metropolitan Complex which indeed includes parts of the Atlantic coast. This explains the occurrence of municiples and postcodes I did not expect, as well as the "seamark" tags, which I now understand are from the Santos Seaboard.

It also explains the occurrence of certain unexpected postcodes:

In [93]:

```
sp.find({'address.postcode' : {'$regex' : '^1[2-9]'}}).count()
```

Out[93]:

```
226
```

In [94]:

```
sp.find_one({'address.postcode' : {'$regex' : '^1[2-9]'}})
```

Out[94]:

```
{u'_id': ObjectId('5734c0c19a07ba122626c129'),
 u'address': {u'city': u'Jundia\xed',
  u'housenumber': u'865',
  u'postcode': u'13201-905',
  u'street': u'Rua XV de Novembro',
  u'suburb': u'Centro'},
 u'amenity': u'hospital',
 u'created': {u'changeset': u'28654626',
  u'timestamp': u'2015-02-06T15:45:48Z',
  u'uid': u'1799626',
  u'user': u'AjBelnuovo',
  u'version': u'2'},
 u'data_prim': u'node',
 u'emergency': u'yes',
 u'id': u'677073968',
 u'name': u'Hospital Paulo Sacramento',
 u'pos': [-23.1890489, -46.8788723]}
```

2. Postcode Format Inconsistencies

Using the '$regex' operator, I was able to audit how postcodes format. By Brazilian convention, the format we should see is 'ddddd-ddd', or the regex ```'^([0-9]){5}([-])([0-9]){3}$'```.

In [7]:

```
sp.find({'address.postcode' : {'$exists' : 1}}).count()
```

Out[7]:

```
9187
```

In [8]:

```
sp.find({'address.postcode' : {'$regex' : '^([0-9]){5}([-])([0-9]){3}$'}}).count()
```

Out[8]:

```
8860
```

The query shows there are some inconsistencies. I want to peek at 10 examples and see some cases.

In [9]:

```
pipe = [{'$match' : {'address.postcode' : { '$regex' : '^(?!^^([0-9]){5}([-])([0-9]){3}$).*$'}}},
        { '$limit' : 10 },
        {'$project' : {'address' : 1 }}]
list(sp.aggregate(pipe))
```

Out[9]:

```
[{u'_id': ObjectId('573655fd9a07ba122647b066'),
  u'address': {u'city': u'S\xe3o Paulo',
   u'postcode': u'010196-200',
   u'street': u'Pra\xe7a do Carmo',
   u'suburb': u'S\xe9'}},
 {u'_id': ObjectId('573655fe9a07ba122647ea6f'),
  u'address': {u'housenumber': u'2022',
   u'postcode': u'04345000',
   u'street': u'Avenida Engenheiro Armando de Arruda Pereira'}},
 {u'_id': ObjectId('5736560d9a07ba12264ca6bd'),
  u'address': {u'housenumber': u'5445',
   u'postcode': u'12315280',
   u'street': u'Avenida Juscelino Kubitschek'}},
 {u'_id': ObjectId('5736560f9a07ba12264d402b'),
  u'address': {u'housenumber': u'137',
   u'postcode': u'0454-000',
   u'street': u'Rua Gra\xfana'}},
 {u'_id': ObjectId('5736560f9a07ba12264d4e89'),
  u'address': {u'housename': u'BurtiHD',
   u'housenumber': u'974',
   u'postcode': u'05311000',
   u'street': u'Avenida Mofarrej'}},
 {u'_id': ObjectId('573656169a07ba12264f95f8'),
  u'address': {u'housenumber': u'162',
   u'postcode': u'01309000',
   u'street': u'Rua Lu\xeds Coelho'}},
 {u'_id': ObjectId('573656169a07ba12264f95f9'),
  u'address': {u'housenumber': u'806',
   u'postcode': u'05006000',
   u'street': u'Rua Turiass\xfa'}},
 {u'_id': ObjectId('573656169a07ba12264f95fa'),
  u'address': {u'housenumber': u'404',
   u'postcode': u'01309010',
   u'street': u'Rua Ant\xf4nio Carlos'}},
 {u'_id': ObjectId('573656179a07ba12264fb2fc'),
  u'address': {u'housenumber': u'744',
   u'postcode': u'01304001',
   u'street': u'Rua Augusta'}},
 {u'_id': ObjectId('573656199a07ba12265072b4'),
  u'address': {u'housename': u"Condom\xednio Edif\xedcio OPUS D'ART",
   u'housenumber': u'905',
   u'postcode': u'05025010',
   u'street': u'Rua Raul Pomp\xe9ia, 905'}}]
```

It seems there is a mix of incorrect format, such as '05025010' instead of '05025-010' and typos like an extra number or a missing one. My solution is in the first case, reformat, the second, discard. This was included in **shape_element**, and the data re-parsed and re-loaded into MongoDB.

In [23]:

```
pipe = [{'$match' : {'address.postcode' : { '$regex' : '^(?!^^([0-9]){5}([-])([0-9]){3}$).*$'}}},
        { '$limit' : 10 },
        {'$project' : {'address' : 1 }}]
list(sp.aggregate(pipe))
```

Out[23]:

```
[]
```

As seen above, a query for 10 postcodes that do not fit the format now returns an empty list, showing the problem was fixed.

OBS: A similar process *can* be done for phone numbers, however it is an intricate process for brazilian phone numbers are there are many different formats. (At this point numbers can have 8 to 10 digits excluding area code, there is an optional designation of operator, and other complicating issues.) This will require a little more time consuming research.

3. Incorrect Postcode

In [92]:

```
t = sp.find({'address.postcode' : {'$regex' : '^2'}})
pprint.pprint(list(t))
```

```
[{u'_id': ObjectId('5734c1239a07ba122644a8aa'),
  u'address': {u'city': u'S\xe3o Paulo',
               u'housenumber': u'456',
               u'postcode': u'25450-000',
               u'street': u'Rua Zilda',
               u'suburb': u'Casa Verde'},
  u'amenity': u'pharmacy',
  u'building': u'yes',
  u'created': {u'changeset': u'28762991',
               u'timestamp': u'2015-02-10T23:31:28Z',
               u'uid': u'1799626',
               u'user': u'AjBelnuovo',
               u'version': u'2'},
  u'data_prim': u'way',
  u'id': u'327507941',
  u'name': u'BiFarma',
  u'node_refs': [u'3342962713',
                 u'3342962714',
                 u'3342962715',
                 u'3342962716',
                 u'3342962713'],
  u'phone': u'+55 11 3857 4511'}]
```

The above postcode is incorrect, it is supposed to be 02545-000.

4. Missing Postcodes (CEPs)

In [2]:

```
a = sp.find({'address' : {'$exists' : 1}}).count()
b = sp.find({'address' : {'$exists' : 1}, 'address.postcode' : {'$exists' : 1}}).count()
c = sp.find({'address' : {'$exists' : 1}, 'address.postcode' : {'$exists' : 0}}).count()
print 'number of addresses:', a
print 'number of addresses with CEP:', b
print 'number of addresses without CEP:', c
print 'percentage of addresses missing CEP:',int((float(c)/float(a))*100),'%'
```

```
number of addresses: 18224
number of addresses with CEP: 9187
number of addresses without CEP: 9037
percentage of addresses missing CEP: 49 %
```

Almost half of the addresses do not have postcodes (called "CEP" in Brazil). One follow-up project would be to scrape the CEPs from a reputable website (like Correios) and feed the CEPs back into the database. A good measure would be to obtain them by coordinates and by address both and analyse discrepancies.

Statistical Overview of the Data

This section contains statistical facts about the data, as well as the query used to determine it when applicable.

XML file size:

411,798 KB

JSON file:

is 472,242 KB

number of documents in the colection:

In [11]:

```
sp.find().count()
```

Out[11]:

```
2168319
```

number of documents by data primitive:

In [12]:

```
cursor = sp.find({'data_prim' : 'way'})
a = len(list(cursor))
print a, 'ways'
cursor = sp.find({'data_prim' : 'node'})
b = len(list(cursor))
print b, 'nodes'
cursor = sp.find({'data_prim' : 'relation'})
c =  len(list(cursor))
print c, 'relations'
```

```
261695 ways
1900322 nodes
6302 relations
```

unique user ids:

In [20]:

```
len(sp.distinct('created.uid'))
```

Out[20]:

```
1747
```

the document that has the highest number of versions:

In [25]:

```
versions = sp.distinct('created.version')
print max(versions)
pprint.pprint(list(sp.find({'created.version' : max(versions)})))
```

```
445
[{u'ISO3166-1': {u'numeric': u'076'},
  u'_id': ObjectId('573656729a07ba122666c255'),
  u'admin_level': u'2',
  u'alt_name': {u'vi': u'Brazil;Bra-xin'},
  u'boundary': u'administrative',
  u'created': {u'changeset': u'38321421',
               u'timestamp': u'2016-04-05T15:33:47Z',
               u'uid': u'1852029',
               u'user': u'smaprs',
               u'version': 445},
  u'currency': u'BRL',
  u'data_prim': u'relation',
  u'driving_side': u'right',
  u'flag': u'http://upload.wikimedia.org/wikipedia/commons/0/05/Flag_of_Brazil.svg',
  u'id': u'59470',
  u'name': {u'zh-classical': u'\u5df4\u897f'},
  u'official_name': {u'zh': u'\u5df4\u897f\u8054\u90a6\u5171\u548c\u56fd'},
  u'old_name': {u'vi': u'Ba T\xe2y'},
  u'type': u'boundary',
  u'wikidata': u'Q155',
  u'wikipedia': u'pt:Brasil'}]
```

'amenities' that occur the most, top 10

In [35]:

```
pipe = [{'$match' : {'amenity': {'$exists' : 1}}},
        {'$group': {'_id': '$amenity', 'count': {'$sum': 1}}},
        {'$sort' : {'count': -1}},
        {'$limit' : 10},
        {'$project' : {'amenity' : 1, 'count': 1}}
        ]
c = sp.aggregate(pipe)
pprint.pprint(list(c))
```

```
[{u'_id': u'fuel', u'count': 1626},
 {u'_id': u'parking', u'count': 1231},
 {u'_id': u'restaurant', u'count': 997},
 {u'_id': u'school', u'count': 893},
 {u'_id': u'bank', u'count': 837},
 {u'_id': u'fast_food', u'count': 589},
 {u'_id': u'place_of_worship', u'count': 519},
 {u'_id': u'pharmacy', u'count': 384},
 {u'_id': u'pub', u'count': 310},
 {u'_id': u'bicycle_rental', u'count': 276}]
```

top 10 religions

In [37]:

```
pipe = [{'$match' : {'amenity': 'place_of_worship'}},
        {'$group': {'_id': '$religion', 'count': {'$sum': 1}}},
        {'$sort' : {'count': -1}},
        {'$limit' : 10},
        {'$project' : {'religion' : 1, 'count': 1}}
        ]
c = sp.aggregate(pipe)
pprint.pprint(list(c))
```

```
[{u'_id': u'christian', u'count': 442},
 {u'_id': None, u'count': 56},
 {u'_id': u'buddhist', u'count': 6},
 {u'_id': u'muslim', u'count': 4},
 {u'_id': u'spiritualist', u'count': 3},
 {u'_id': u'jewish', u'count': 2},
 {u'_id': u'umbanda', u'count': 1},
 {u'_id': u'candombl\xe9_na\xe7\xe3o_gege', u'count': 1},
 {u'_id': u'multifaith', u'count': 1},
 {u'_id': u'candombl\xe9', u'count': 1}]
```

This shows this data is severely incomplete and does not lend itself for statistics, not on religion, anyway. There are lenty more than 2 Jewish places of worship in Sao Paulo.

number of pizza places:

In [39]:

```
sp.find({'cuisine': 'pizza'}).count()
```

Out[39]:

```
100
```

Again, I am sure that it actually is more.

Examples of Curiosity Queries

This section contains queries that were performed motivated by personal curiosity. If the choices seem arbitrary, it is because they are.

How many street names have a military rank in the name?

In [49]:

```
def street_starts_with(letters):
    """takes a string and returns a regex string to be used with operator
    $regex to query sp collections for streets starting with the string"""
    a = ['Acostamento',
         u'Pra\xe7a',
         'Alameda',
         'Viela',
         'Estrada',
         'Rua',
         'Acesso',
         'Parque',
         'Largo',
         'Via',
         'Marginal',
         'Rodovia',
         'Corredor',
         'Viaduto',
         'Travessa',
         'Pateo',
         'Avenida',
         'Passagem',
         u'Complexo Vi\xe1rio']
    expression = '|'.join(a)
    return '^'+'(' + expression + ')' + ' ' + letters

""" from wikipedia: Almirante Marechal Marechal do Ar
Almirante de Esquadra General de Exército Tenente Brigadeiro do Ar
Vice Almirante General de Divisão Major Brigadeiro
Contra Almirante General de Brigada Brigadeiro
Capitão de Mar e Guerra Coronel Coronel
Capitão de Fragata Tenente Coronel Tenente Coronel
Capitão de Corveta Major Major
Capitão Tenente Capitão Capitão """

military_ranks = ['Almirante',
                  'Marechal',
                  'Marechal',
                  'General',
                  'Tenente',
                  'Brigadeiro',
                  'Major',
                  'Contra Almirante',
                  u'Capitão']
nomes = {}
for rank in military_ranks:
    nomes[rank] = set()
    result =  sp.find({'data_prim' : 'way', 'name': {'$regex': street_starts_with(rank)}})
    for r in result:
        nomes[rank].add(r['name'])
soma = 0
for k in nomes:
    soma += len(nomes[k])
    print k, ':', len(nomes[k])
print 'TOTAL:', soma
```

```
Major : 75
Capitão : 154
Marechal : 90
Tenente : 89
General : 195
Almirante : 49
Brigadeiro : 34
Contra Almirante : 1
TOTAL: 687
```

How many street names start with X? And Z?

In [54]:

```
x = sp.find({'data_prim':'way', 'name': {'$regex': street_starts_with('X')}}).count()
print 'Starting with X:', x
z = sp.find({'data_prim':'way', 'name': {'$regex': street_starts_with('Z')}}).count()
print 'Starting with Z:', z
```

```
Starting with X: 148
Starting with Z: 195
```

Conclusion

There are some unanswered questions that could lend themselves to further projects:

- **What is the exact delimitations of the data and is it complying with official government designation for the Expanded Metropolitan Complex?**

It proved harder than it seems to obtain such official designation. I fully intend to continue to pursue this question. Once the official information is at hand, it will not be hard to check against the data.

- **How many street, schools or other public places have been named after officials of the military dictatorship under which Brazil was governed between 1964 and 1985?**

This is a bigger project, requiring the scraping of quantities of information on the historical period. It is, however, feasible and an important question to be answered. Perhaps, initially it can be answered for the city only and then expand the question.

Overall, the data lacks completeness and normalization. One important task is to add the postcodes to all the addresses, a plan for which is outlined above. Another idea for data validation is to check the data for consistency with information scraped from Correios website, which contains the official post office information in Brazil. Once the post office data is understood and one is able to perform efficient requests and scraping, the following validation tests can be performed:

- postcodes from data vs POS
- postcode from data vs street names

This is powerful data and can be used in many different creative ways, such as using the POS information to make plots showing what is popular in each neighborhood, which ones have more parks, different types of churches. The possibilities are endless once the data is properly clean, normalized and validated.

References

- http://www.openstreetmap.org/relation/2661855#map=9/-23.6242/-46.4510
- several pages os OSM Wiki
- https://pt.wikipedia.org/wiki/Regi%C3%A3o_Metropolitana_da_Baixada_Santista
- https://pt.wikipedia.org/wiki/Complexo_Metropolitano_Expandido
- https://pt.wikipedia.org/wiki/Regi%C3%A3o_Metropolitana_de_S%C3%A3o_Paulo#Munic.C3.ADpios
- https://en.wikipedia.org/wiki/Expanded_Metropolitan_Complex_of_S%C3%A3o_Paulo
- https://pt.wikipedia.org/wiki/Complexo_Metropolitano_Expandido
- http://stackoverflow.com/questions/37014500/how-to-use-recursion-to-nest-dictionaries-while-integrating-with-existing-record
- https://discussions.udacity.com/t/reducing-memory-footprint-when-processing-large-datasets-in-xml/37571/3
- http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-file-generated-via-xml-etree-elementtree-in-python
- https://discussions.udacity.com/t/how-to-provide-sample-data-for-the-final-project/7118/13
- https://discussions.udacity.com/t/valueerror-i-o-operation-on-closed-file/167469/6
- https://discussions.udacity.com/t/keep-attr-atrr-atrr-formatted-data/166864/14
- https://discussions.udacity.com/t/i-have-an-adequate-update-name-to-improve-street-names-not-sure-how-to-use-it/166569/5