Rubric Questions:

1. Problems Encountered in the Map

Inconsistent abbreviations

Before I imported the data to MongoDB I took a quick look at the street names with the following code:

```
import re
import xml.etree.cElementTree as ET
import pprint
from collections import defaultdict
import codecs
import json
from pymongo import MongoClient

#filename = "sample.osm"
filename = "san-diego-tijuana_mexico.osm"

street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons"]

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

def audit_street_type(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)

def check_street_names():
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(filename, events = ("start",)):
        if elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    pprint.pprint(dict(street_types))


check_street_names()
```

The code groups the different street types. The output showed that for some street types there were several inconsistent abbreviations (e.g. “Av”, “Ave” and “Ave.” for Avenue). I don’t mind abbreviations in general, but I decided to remove the inconsistencies (e.g. replace “Av” and “Ave.” with “Ave”). The changes are done according to the following key-value pairs:

```
mapping = {
        r"St.\b": "St",
        r"\bBl\b" : "Blvd",
        r"Blvd.\b" : "Blvd",
        r"Ave.\b":"Ave",
        r"Av\b" : "Ave",
        r"Rd.\b":"Rd",
        r"Py\b" : "Pkwy",
        r"Prky\b" : "Pkwy",
        r"Wy\b": "Way",
        r”\bstreet\b" : "Street"
        }
```

A given street name can be updated using regular expressions as follows:

```
def update_name(name, mapping):
    for key in mapping:
        name = re.sub(key,mapping[key],name)
    return name
```

Specifying the data model and updating the street names

The updates of the street names was performed during the data conversion process, i.e. transforming the OSM/XML-file into a JSON/Python dict format:

```
CREATED = [ "version", "changeset", "timestamp", "user", "uid"]

#Convert data into dict format and save as a JSON file
def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :

        node['created'] = {}
        la = None
        lo = None
        for at in element.attrib:
            if at == 'lat':
                la = float(element.attrib[at])
            elif at == 'lon':
                lo = float(element.attrib[at])
            elif at in CREATED:
                    node['created'][at] = element.attrib[at]
```

```
            else:
                node[at] = element.attrib[at]

        if la!= None:    #i.e. latitude information is actually present
            node['pos'] = [la,lo]

        node['address'] = {}
        for tag in element.iter("tag"):
            k_split = tag.attrib['k'].split(":")
            if k_split[0] == 'addr':
                if len(k_split) > 2:
                    continue
                else:
                    if k_split[1] == "street":              #key == “addr:street"
                                                            #if necessary update name according to 'mapping'
                        node['address'][k_split[1]] = update_name(tag.attrib['v'],mapping)

                    else:                                   #different key (not a street name)
                        node['address'][k_split[1]] = tag.attrib['v']
            else:
                node[tag.attrib['k']] = tag.attrib[‘v’]

        #check for empty fields and delete them
        if node['address'] == {}:
            del node['address']
        if node['created'] == {}:
            del node['created']

        node['node_refs'] = []
        for nd in element.iter("nd"):
            node['node_refs'].append(nd.attrib['ref'])
        if node['node_refs'] == []:
            del node[‘node_refs']

        node['type'] = element.tag #this possibly overrides already existing tags with key “type”
                                   #the line has moved to the bottom as in a preliminary analysis
                                   #556 such tags were found using the MongoDB query:
                                   #db.data.find({“$and":[{"type":{"$ne":"way"}},{"type":
                                   #{“$ne":"node"}}]}).count()

        return node
    else:
        return None


def process_map(file_in):
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
```

```
    for _, element in ET.iterparse(file_in):
        el = shape_element(element)
        if el:
            data.append(el)
            fo.write(json.dumps(el) + "\n")
    return data

process_map(filename)
```

Non numeric postcodes

I found a few ZIP codes that were badly formatted. To make it more interesting (and since we are dealing with only a few values) the cleaning of the ZIP codes will be performed after importing the data into the MongoDB database. This gives us the chance to explore the MongoDB regular expression capabilities. See “More cleaning” in Section 2.

Problems particular to the region could not be found.

2. Data Overview

Import to MongoDB

I "bulk" imported the created JSON file using the shell command ‘*mongoimport*’:

mongoimport -d sd -c data --file san-diego-tijuana_mexico.osm.json --drop

$.....imported 1594251 documents

Size

Size of the OSM file: 548 MB
Size of the converted in imported JSON file: 517.4 MB

Overview

```
from pymongo import MongoClient

def get_db(dbase):
```

```
    client = MongoClient('localhost:27017')
    db = client[dbase]
    return db

#Access database
db = get_db("sd")

#Number of files
db.data.find().count() #1594251

#Number of nodes
db.data.find({"type":"node"}).count() #1480433

#Number of ways
db.data.find({"type":"way"}).count() #113818

# Top 1 contributing user
result = db.data.aggregate([{"$group":{"_id":"$created.user","count":{"$sum":1}}},{"$sort":
{"count":-1}},{"$limit":1}])
print list(result) #[{u'count': 684272, u'_id': u'Adam Geitgey'}]

# Number of users appearing only once (having 1 post)
result2 = db.data.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}}, {"$group":
{"_id":"$count", "num_users":{"$sum":1}}}, {"$sort":{"_id":1}}, {"$limit":1}])
print list(result2) #[{u'num_users': 189, u'_id': 1}]
```

More cleaning

First we find all elements with a ZIP code (part of the address information):

```
postcodes = db.data.find({“address.postcode":{"$exists":True}})
```

We are particularly interesting in finding postcodes that contain non-numerical values. We make use of MongoDB’s regular expression capabilities using the *$regex* command:

```
postcodes_non_numeric = postcodes.collection.find({"address.postcode":{"$regex":"[^0-9]"}})
postcodes_non_numeric.count() #47

#Print the result
for a in postcodes_non_numeric:
    print a["address"]["postcode"]
```

CA 92109
92010-1407
92071-4417
92108-3803
92103-3607

92103-3609
92103-3609
CA 92109
CA 92109
CA 92109
CA 92101
CA 92101
CA 92106
CA 92101
CA 92101
CA 92101
CA 92101
CA 92101
CA 92121
CA 92121
CA 92110
CA 92110
92110-9998
92093-0094
CA 91901
CA 92101
CA 92101-6144 #(check result below)
CA 92101
92093-0068
CA 92101
San Diego, CA
CA 92128
CA 92128
92102-4810
92111-2201
CA 91914
Scripps Ranch Blvd.
CA 92101
CA 92101
CA 92101
CA 92101
CA 92101
CA 92101
CA 92101
CA 92101

We see three problems:

I. Postodes that start with “CA ”
II. “ZIP +4 Codes”
III. City or street names

We tackle these cases as follows:

ad I. Separate CA and update address.state
ad II. Separate the last 4 digits and create a new tag “postbox”

ad III. Delete tag

```
#Split if there is a +4 ZIP code:
for a in postcodes_non_numeric:
    x = re.split("-",a["address"]["postcode"])
    a["address"]["postcode"] = x[0]
    if len(x) > 1:
        a["address"]["postbox"] = x[1] #create new tag
    db.data.save(a)

postcodes_non_numeric = postcodes.collection.find({“address.postcode":{"$regex":"[^0-9]"}})

#Split remaining ZIP codes and check for “CA”:
for a in postcodes_non_numeric:
    x = re.split(" ",a["address"]["postcode"])
    if  x[0] == "CA":
        a["address"]["state"] = "CA"
        a["address"]["postcode"] = x[1]
    else:
        del a["address"]["postcode"] #delete others
    db.data.save(a)

postcodes.collection.find({"address.postcode":{"$regex":"[^0-9]"}}).count() #0
db.data.find_one({“address.postbox":{"$exists":True}})

Out[177]:
{u'_id': ObjectId('554907242b65e2072153604b'),
 u'address': {u'city': u'San Diego',
  u'housenumber': u'79',
  u'postbox': u'6144',
  u'postcode': u'92101',
  u'state': u'CA',
  u'street': u'Horton Plaza'},
 u'amenity': u'theatre',
 u'created': {u'changeset': u'20328417',
  u'timestamp': u'2014-02-02T05:07:18Z',
  u'uid': u'945299',
  u'user': u'Trekoid',
  u'version': u'1'},
 u'id': u'2649190809',
 u'name': u'Lyceum Theater',
 u'operator': u'San Diego Rep',
 u'phone': u'+1 619 544 1000',
 u'pos': [32.714595, -117.161877],
 u'type': u'node',
 u'website': u'http://lyceumevents.org/'}
```

Overview of restaurants

#Number of restaurants
db.data.find({"amenity":"restaurant"}).count() #643
db.data.find({"$and":[{"amenity":"restaurant"},{"type":"node"}]}).count() #544
db.data.find({"$and":[{"amenity":"restaurant"},{"type":"way"}]}).count() #99

#Number of restaurants with (partial) addresss information
db.data.find({"$and":[{"amenity":"restaurant"},{"address": {"$exists": True}}]}).count() #160

#Number of restaurants with a given country code
db.data.find({"$and":[{"amenity":"restaurant"},{"is_in:country_code": {"$exists": True}}]}).count()
#48

This tells us that many restaurants miss information about their location, which gives raise to the following idea.

3. Additional Ideas

As the dataset is dealing with a metropolitan area that covers two countries, for a lot of restaurants there would be no explicit tag from which could be inferred directly whether the restaurant would be located in Mexico or the U.S.. For only 48 restaurants such a tag ("is_in:country_code") given.
As we are talking about OSM data, every element should contain information about it's location (given as a tuple "pos" : [latitude, longitude]) which can be confirmed like

restaurants = db.data.find({"$and":[{"amenity":"restaurant"},{"pos": {"$exists": True}}]})
restaurants.count()

which gives 544 (= total number of restaurant nodes).

Therefore, the idea is to update the "is_in:country_code" for all restaurants using the location information in the "pos" field. To do so, I approximate the border as a straight line.

#The border Mexico - USA through San Diego and Tijuana follows approximately this simple straight line where x represents latitude and y longitude:
#y = 0.0906502699248 * (x + 117.124002) + 32.533842

```
def us_border(pos):
    x,y = pos #unpack the position array
    if (0.0906502699248 * (x + 117.124002) +  32.533842) < y:
        return False #if y below the border line => on the Mexican side
    else:
        return True # above => U.S.
```

#update "is_in:country_code" by looping through all restaurants

```
for r in restaurants:
    if us_border(r["pos"]):
        r["is_in:country_code"] = "US"
    else:
        r["is_in:country_code"] = "MX"
    db.data.save(r)

db.data.find_one({"$and":[{"amenity":"restaurant"},{"address": {"$exists": True}}]})
```

```
Out[155]:
{u'is_in:country_code': u'US',
 u'_id': ObjectId('554673422b65e207212b891f'),
 u'address': {u'city': u'San Diego',
  u'country': u'US',
  u'housenumber': u'4110',
  u'postcode': u'92105',
  u'street': u'Home Ave'},
 u'alt_name': u'Chiquitas',
 u'amenity': u'restaurant',
 u'contact:phone': u'619.264.2072',
 u'created': {u'changeset': u'11503119',
  u'timestamp': u'2012-05-04T23:16:26Z',
  u'uid': u'674368',
  u'user': u'Ron Larson',
  u'version': u'8'},
 u'cuisine': u'mexican',
 u'drive_in': u'no',
 u'id': u'356140998',
 u'name': u"Chiquita's Mexican Food",
 u'pos': [32.723789, -117.1070615],
 u'smoking': u'no',
 u'source': u'Personal knowledge and visit.',
 u'takeaway': u'yes',
 u'type': u'node'}
```

As we can see, the “is_in:country_code” field has been updated correctly. In this case, there already has been address information. However, I wanted to try something new and explore how to update MongoDB collections.

TIGER data

During the cleaning I found a bunch of tags named “tiger”. A quick glance at the OSM wiki [4)] is data from the US Census Bureau. The TIGER (**T**opologically **I**ntegrated **G**eographic **E**ncoding and **R**eferencing system) data is of varying quality.
To see how many data points in our dataset have not yet been review, we can use the tag “tiger:reviewed”:

tiger = db.data.find({“tiger:reviewed":"no"})
print tiger.count() #30341

30,341 files stemming from TIGER data have not yet been reviewed.

Python Code for Lesson 6 quizzes:

See attachment.

Link to the Map:

File: https://s3.amazonaws.com/metro-extracts.mapzen.com/san-diego-tijuana_mexico.osm.bz2

Position: http://www.openstreetmap.org/#map=10/32.8254/-117.2791

I chose San Diego because I will move there this summer.

Sample file:

See attachment.

References:

1) https://wiki.openstreetmap.org/wiki/OSM_XML#Example_OSM_XML_file

2) http://docs.mongodb.org/manual/reference/

3) http://api.mongodb.org/python/current/examples/aggregation.html

4) http://www.regular-expressions.info/quickstart.html

5) http://wiki.openstreetmap.org/wiki/TIGER