OpenStreetMap Project
Data Wrangling with MongoDB

Map Area: Las Vegas, NV, United States

Student: Bingson Huang

CONTENTS

I. PROJECT QUESTIONS

A. PROBLEMS ENCOUNTERED IN THE MAP

To narrow the scope of the discussion, this section focusses on problems endemic to address related fields in Las Vegas Open Street Map (OSM) XML data. It's important to note that the following functions must be called in a specific order — i.e., path dependent logic.

1. Inconsistent capitalization in addr:street, addr:housename, and addr:city values

Problem and cleaned data examples:

[update_street] W PACIFIC AVE -> West Pacific Avenue
[update_housename] hiro karate -> Hiro Karate
[update_city] las vegas -> Las Vegas

Example cleaning code:

```
exceptions = ['MMS', 'NV', 'AZ', 'a', 'an', 'of', 'the', 'is', 'AFB', 'RTC', 'US']
s = input string

def capitalize_first_letter(s, exceptions):
    word_list = re.split(' ', s)
    formatted_word_list = []
    for i in word_list:
        try:
            if i in exceptions or i[0].isdigit(): formatted_word_list.append(i)
              # i[0].isdigit() -> exclude '3rd' or '4th' street elements
            else: formatted_word_list.append(i[0].upper() + i[1:].lower())
        except: continue
    return " ".join(formatted_word_list)
```

- The cleaning function capitalizes the first letter of each word in the input string (while other letters are converted to lower case) as long as the word does not appear in the exceptions array. Words that appear in the exceptions array will remain unaltered.
- The capitalize function is an early part of the 'addr:street', 'addr:city', and 'addr:housename' cleaning process because it reduces the number of cases that must be considered for subsequent string substitution functions described further below.
- Special cases and challenges (all solved):

  3rd Street — '3rd' should not be changed to '3Rd'
  Nevada Department of Motor Vehicles — 'of' should remain in lower case
  MMS Scooters — 'MMS' should remain capitalized, not transformed to 'Mms'

2. Inconsistent abbreviations in 'addr:street' and 'addr:state'

Problem and cleaned data examples:

[update_state] Nevada -> NV
[update_street] W Sahara Ave -> West Sahara Avenue
[update_street] Thomas Ryan Blvd -> Thomas Ryan Boulevard
[update_street] El Camino Rd -> El Camino Road
[update_street] S Eastern Ave #100 -> South Eastern Avenue

Example cleaning code :

```
s_pattern = re.compile(r'\s?(?<!\S)S\.?\s|\s(?<!\S)S\.?$')
address_buffer = s_pattern.sub(r' South ', address_buffer)
# similar search and replace operations were used to standardize the formatting and presentation of
# several address related fields
```

- Instead of writing lengthy blocks of 'If…Then' or 'for loops' to find and modify string sequences, I opted for a more flexible regular expression-based approach. Regular expressions makes it easier to adapt cleaning functions to new constraints or different formatting conventions (e.g., different city map) by changing pattern match specifications illustrated above.

3. Suite and building numbers in the addr:street and addr:housename field

Often suite and building numbers were incorrectly entered into the street field. For uniformity, I moved this data into new address sub fields.

Problem and cleaned data examples:

(new suite number) 2230 Corporate Circle Suite 250 -> 250
(new address number) 2230 Corporate Circle -> 2230
[update_street] 2230 Corporate Circle Suite 250 -> Corporate Circle

Example cleaning code:

```
apt_num = re.compile(r'(Suite|Ste)\s?([-0-9A-Z]+)')
addr_num = re.compile(r'([0-9]+)[^a-z]\s?')

def capture_numbers(string, node):
    string = string.strip()
    if re.search(apt_num, string):
        orig1 = string
        apt_data = re.search(apt_num, string).group()
        string = string.replace(apt_data, '')
        string = string.strip('.,# ')
        clean_apt_num = apt_data.strip('Suite')
        node['address'].update({'captured_suite_num' : clean_apt_num})
        print "(new suite number) " + orig1 + " -> " + clean_apt_num
    if re.search(addr_num, string):
        orig2 = string
        if "Highway" not in string:
            streetnumber = re.search(addr_num, string).group().strip()
            node['address'].update({'captured_addr_num' : streetnumber})
            string = string.replace(streetnumber, '')
            string = string.strip('#.,')
            print "(new address number) " + orig2 + " -> " + streetnumber
            return string, node
    else:
        return string, node
```

- The function above creates new address key/value pairs if it detects suite or building numbers in the input string:
    - captured_suite_num (for numbers following 'Suite' or 'Ste')
    - captured_addr_num (for any other numbers matching the criteria of the search pattern)
- The capitalize_first_letter function was applied to data before applying the above function to reduce the number of regular expression match cases that must be

considered. For example, instead of trying to find numbers immediately following Suite, suite, SUITE, ste, STE, and Ste, after applying the capitalize_first_letter function, my capture_numbers function only has to look for numbers following 'Suite' or 'Ste'.

4. Inconsistent location of direction prefixes

Problem and cleaned data examples:

[update_street] Las Vegas Boulevard South -> South Las Vegas Boulevard
[update_street] Las Vegas Boulevard North -> North Las Vegas Boulevard

Example cleaning code:

```
def move_direction(string):
    if re.search(east_pattern, string): string = "East " + east_pattern.sub(r'', string)
    if re.search(west_pattern, string): string = "West " + west_pattern.sub(r'', string)
    if re.search(south_pattern, string): string = "South " + south_pattern.sub(r'', string)
    if re.search(north_pattern, string): string = "North " + north_pattern.sub(r'', string)
    else: return string
    return string
```

- All direction prefixes (North, South, East, West) were moved to the beginning of the string
- Regular expressions were modified to exclude strings that contained substring matches such as "East" in "Eastern"

  E.g., S. Eastern Ave -> South Eastern Avenue.

5. Duplicate data in fields

Problem and cleaned data examples:

(new address number) 302 E Carson -> 302
[update_housenumber] 302 -> *delete duplicate captured_addr_num
[update_housename] East Carson -> *del duplicate street value in housename

Example cleaning code:

```
if cleaned_data == node['address']['captured_addr_num']:
    del node['address']['captured_addr_num']
```

- similar node deletion commands were applied throughout my cleaning file to reduce the odds of importing redundant or mislabeled data into my json file.

6. State data in postal code field

Problem and cleaned data examples:

(new state) NV 89142 -> NV
[update_postcode] NV 89142 -> 89142

Example cleaning code:

```
def update_state(state_string, node):
    cleaned_state_string =  state_string.upper().replace('NEVADA', 'NV')
    if cleaned_state_string != state_string:
        print "[update_state] "+ state_string + " -> "+ cleaned_state_string
    return cleaned_state_string, node
```

```
def update_state_second_source(tgt_string, node):
    orig = tgt_string
    if re.search(state_name, tgt_string):
        state_label = re.search(state_name, tgt_string).group()
        tgt_string = tgt_string.replace(state_label, '')
        node['address'].update({'state' : 'NV'})
        print "(new state) " + orig + " ->  NV"
          return tgt_string, node
```

- State information incorrectly entered in the postal code field was captured and reassigned.
- Used regular expression pattern to identify postal code data.

B. DATA OVERVIEW

This section contains basic statistics about the dataset and the MongoDB queries used to gather them. The file 3_mongo_queries.py in the accompanying zip file contains the code for the following queries.

File sizes: las-vegas_nevada.osm **180,732 kb**
las-vegas_nevada.osm.json **258,095 kb**

Number of documents: **916423**

```
db.vegas.find().count()
```

Number of nodes: **826675**

```
db.vegas.find({"type" : "node"}).count()
```

Number of ways: **89748**

```
db.vegas.find({"type" : "way"}).count()
```

Number of unique uers: **725**

```
len(db.vegas.distinct("created.user"))
```

Top contributing user: **{u'_id': u'alimamo', u'count': 254361}**

Sample Code

```
def top_contributing_user_pipe():
    pipeline = [
        {"$group" : {"_id" : "$created.user",
                     "count" : { "$sum" : 1}}},
        {"$match" : {"_id" : {"$ne" : None}}},
        {"$sort" :{"count" : -1}},
        {"$limit" : 1}
        ]
    return pipeline

result = db.vegas.aggregate(top_contributing_user_pipe())
```

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

{u'_id': 1, u'num_users': 142}

Sample Code

```
def num_one_time_users_pipe():
    pipeline = [
        {"$group": { "_id" : "$created.user",
                     "count" : {"$sum":1}}},
        {"$group": { "_id" : "$count",
                "num_users" : {"$sum":1}}},
        {"$sort" : { "_id" : 1}},
        {"$limit": 1 }
        ]
    return pipeline
```

Top 20 amenities:

{u'_id': u'parking', u'count': 833}
{u'_id': u'school', u'count': 537}
{u'_id': u'place_of_worship', u'count': 366}
{u'_id': u'fountain', u'count': 270}
{u'_id': u'restaurant', u'count': 225}
{u'_id': u'fast_food', u'count': 190}
{u'_id': u'fuel', u'count': 146}
{u'_id': u'fire_station', u'count': 70}
{u'_id': u'hospital', u'count': 66}
{u'_id': u'toilets', u'count': 65}
{u'_id': u'post_office', u'count': 63}
{u'_id': u'bar', u'count': 56}
{u'_id': u'bank', u'count': 55}
{u'_id': u'cafe', u'count': 53}
{u'_id': u'shelter', u'count': 51}
{u'_id': u'public_building', u'count': 36}
{u'_id': u'casino', u'count': 35}
{u'_id': u'pharmacy', u'count': 27}
{u'_id': u'theatre', u'count': 25}
{u'_id': u'swimming_pool', u'count': 24

Top 10 most convenient dining options:

{u'_id': u'burger', u'count': 69}
{u'_id': u'mexican', u'count': 25}
{u'_id': u'pizza', u'count': 19}
{u'_id': u'coffee_shop', u'count': 16}
{u'_id': u'american', u'count': 16}
{u'_id': u'chicken', u'count': 11}
{u'_id': u'chinese', u'count': 11}
{u'_id': u'sandwich', u'count': 8}
{u'_id': u'italian', u'count': 8}
{u'_id': u'japanese', u'count': 8}

Sample JSON entry:

{'address': {'city': 'Las Vegas',
'country': 'US',
'housenumber': '128',
'postcode': '89101',
'state': 'NV',
'street': 'Fremont Street'},
'amenity': 'casino',
'created': {'changeset': '33345378',
'timestamp': '2015-08-15T01:43:29Z',
'uid': '152074',
'user': 'beweta',
'version': '2'},
'id': '3035637974',
'name': "Binion's Gambling Hall and Hotel",
'phone': '+1 702 382 1600',
'pos': [36.1711359, -115.1442037],
'type': 'node',
'website': 'http://www.binions.com/'}

C. ADDITIONAL IDEAS

While working with the data, I noticed that there were several issues with the completeness of certain entries. For example, there were 35 casinos listed in the original Las Vegas OSM XML file; however, according to the Nevada Gaming Commission, that number should have been closer to 75[1]. This shouldn't be surprising, given the patchwork nature of crowdsourced information. I anticipate that over time, as mapping technology improves and businesses figure out how to monetize OSM data, there will be a more concerted effort by the OSM community to improve data quality.

For example, from the perspective of a national pharmacy chain looking to grow market share, geospatial indexing can improve management's ability to evaluate emerging business risks and assess new expansion opportunities. More specifically, MongoDB already provides support for geospatial indexes, giving Mongo users the ability to run queries for locations near other locations.

At the business strategy level, company management can combine OSM data with government data on neighborhood income levels/trends and internally sourced data on competitor locations or anchor store locations[2] to improve sales forecasts and assess threats to their operations. Moreover, as new commercial/real-estate development opportunities appear on the market, management can use a geospatial filtering algorithm incorporating the above factors to more efficiently identify attractive opportunities.

As alluded to above, the challenge to incorporating geospatially indexed OSM data is that crowd sourced data may not be very reliable or timely. Although certain landmarks and most street names won't change much, information about local businesses would have to be verified regularly (or purchased from a third party vendor), representing an additional out of pocket cost for the company.

1) http://gamboool.com/how-many-casinos-are-on-the-las-vegas-strip

2) anchor stores are stores in an area that attract additional customer traffic such as grocery stores or big box retailers

II. OTHER PROJECT REQUIREMENTS

A. WORKS CONSULTED

1. **Beazley, David and Jones, Brian K.** *Python Cookbook*. s.l. : O'Reilly Media, Inc., 2013. ISBN: 978-1-449-34037-7.
2. **Rodríguez, Gabriel.** Pythex (regular expression tester). [Online] http://pythex.org/.
3. **RegexOne.** RegexOne (regular expression interactive tutorial). [Online] http://regexone.com/.
4. **Open Street Map.** Open Street Map Wiki. [Online] https://wiki.openstreetmap.org/.
5. **Chan, Jamie.** Learn Python in One Day and Learn it Well. s.l. : Learn Coding Fast, 2014. http://www.learncodingfast.com/python.

B. LINK TO MAP POSITION

https://www.openstreetmap.org/node/31551114
https://s3.amazonaws.com/metro-extracts.mapzen.com/las-vegas_nevada.osm.bz2

I chose the Las Vegas because it's a medium sized city famous for its many casinos, providing readily available reference data I can use to check OSM accuracy and completeness.