Clayton Schuler Udacity Data Analyst Nanodegree
2/1/2017

P3: Fort Collins Open Street Map Cleaning

Overview:

fort_collins.osm ……. 63,570 KB
fc_osm.db …….. 35,734 KB

nodes.csv …….. 24,650 KB
nodes_tags.csv ……. 1,243 KB
ways.csv ……. 1,967 KB
ways_nodes.csv …… 8,334 KB
ways_tags.csv …… 4,046 KB

Nodes and Ways Counts

In order to determine numbers of nodes/ways, I used code to query:

```
for table in ["Nodes", "Nodes_Tags", "Ways", "Ways_Nodes", "Ways_Tags"]:
    query = "SELECT count(*) FROM {0}".format(table)
    c.execute(query)
```

Outputting:

Nodes contains 297777 entries.
Nodes_Tags contains 33955 entries.
Ways contains 33671 entries.
Ways_Nodes contains 345642 entries.
Ways_Tags contains 115148 entries.

Problems Encountered:

I used an audit file to parse through and suspect data into sets to be reviewed. These sets were then used to create specific mappings and write a class CleanValue to correct the data.

Street Names

Street names have many technically correct designations for addresses. However, in order to create uniformity and efficiency, I determined to unify all types to a full written street type (ie “Road”, “Street”, “Drive”, etc). Since many of the initial types included acronyms such as “Rd”, “Str”, or “Dr.”, I created a street names mapping called by the Cleaner.

During auditing, my function caught some correct names that I had missed in my expected names, such as "Alley" as well as a direction at the end (eg. "Sheffield Circle West"), so I went back and added these to my expected list.

output of check_streets():

```
238 : set(['Old Town Square #238'])
East : set(['Sheffield Circle East'])
```

Timberline : set(['S Timberline'])
7 : set(['South County Road 7'])
287 : set(['North US Highway 287'])
St : set(['W Willox St', 'East Mulberry St'])
st : set(['3501 stover st'])
Aventue : set(['West Mountain Aventue'])
Alley : set(['Old Firehouse Alley'])
Church : set(['Seventh-Day Adventist Church'])

Many of the "irregular" street types were caught because they end in a number, which is completely valid in their cases, such as highways and county roads.
Some incorrect formats include inconsistent abbreviations used a number of times ("St","st","Rd",etc), misspelling of "Avenue" as "Aventue" and ignoring the type of street altogether such as in "West Oak" (Street) and "S Timberline" (Road).
There also appear to be an address that got mixed into the streets: "Seventh-Day Adventist Church", which is actually on "East Pitkin Street", and "Old Town Square #238", which incorrectly includes a suite number, and should be "Old Town Square". "S Summit View #11" should be "S Summit View Drive".

I checked the child tags for the more difficult issues before manually updating them in my manual mapping for CleanValue.
(example: I would use the following conditional for the iterator, replacing *desired_value* with the specific value I was looking for:
“””
```
for _, elem in ET.iterparse(filepath):
    if elem.tag == "node" or elem.tag == "way":
        children = elem.findall("tag")

        for child in children:
            child_key = child.attrib['k']
            child_val = child.attrib['v']
            if "street" in child_key:
                desired_value = "Seventh-Day Adventist Church"
                if child_val == desired_value:

                    for child in children:
                        print child.attrib
```
“””
Which returned:
{'k': 'name', 'v': 'Seventh-Day Adventist Church'}
{'k': 'amenity', 'v': 'place_of_worship'}
{'k': 'building', 'v': 'church'}
{'k': 'religion', 'v': 'christian'}

{'k': 'addr:city', 'v': 'Fort Collins'}
{'k': 'addr:state', 'v': 'Colorado'}
{'k': 'addr:street', 'v': 'Seventh-Day Adventist Church'}
{'k': 'addr:postcode', 'v': '80524-3817'}
{'k': 'contact:phone', 'v': '+1-970-482-7365'}
{'k': 'contact:website', 'v': 'http://www.fcsdachurch.org/'}
{'k': 'addr:housenumber', 'v': '502'}

With this information, i was easily able to locate the actual street, which I added to my manual mapping to be corrected when parsed.
After being run through CleanValue, this sampling of issues looks more suitable:

S Timberline Road
West Mountain Avenue
Old Firehouse Alley
W Willox Street
East Mulberry Street
Old Town Square
South County Road 7
E Pitkin Street
Ringneck Drive
North US Highway 287

Postal Codes

Postal Codes was a tricky section for correction. Although most codes were input correctly into the open street map, many of them contained an additional set of digits on the end (called a zip + 4). This 4 digits are extremely valuable to include for location. Instead I spliced the zip to check that the first 5 digits are within the correct expectations for the city and surrounding area of Fort Collins.
I did encounter the specific issue of postal codes being formatted incorrectly with the zip being led by the State abbreviation (ie CO 80525). This was easy to map to, when detected, splice out the str, then validate the zip and pass it.
(10 sample flagged postal codes from audit)

80538-1160
80535-9432
80547-4416
80538
80701
CO 80526
80535-9362
80535-9359
80547
80535

Many of the issues I saw were just from local areas being included in the Fort Collins data. This is not an issues and actually adds to the completeness of the map, since there are realistically overlaps in the real world. Therefore I decided to check and add them to expected values.

The most difficult issue I encountered was a zip code of “80701”, which references a very distant city district. By looking at the attributes of this transgressing postal code tag by changing (from the above iteration):

```
if "post" in child_key:
   desired_value = “80701":
```

Outputting:

```
{'k': 'name', 'v': 'CSP Weigh Station: Fort Collins'}
{'k': 'amenity', 'v': 'police'}
{'k': 'building', 'v': 'yes'}
{'k': 'operator', 'v': 'Colorado State Patrol'}
{'k': 'addr:city', 'v': 'Fort Collins'}
{'k': 'addr:state', 'v': 'CO'}
{'k': 'enforcement', 'v': 'weigh-station'}
{'k': 'addr:postcode', 'v': '80701'}
```

It appeared as if this was a mistake from copying a distant weigh station template from Fort Morgan, CO (source of ‘80701’) and only updating the city. The above Weigh Station belongs in the postal district ‘80525’, with all other information correct. Therefore this was again added to a mapping to be called by CleanValue.

Cleaning

Since I predict this is a section that can continually be updated or expanded to include other cleaning issues, I decided to write this code in object oriented form in order to maximize extensibility (class CleanValue()). If I need to update it to also correct, say, Phone Number format for example, I can easily include another CleanValue method with a mapping without worrying about diving back into the code bodies on my primary files.

Analysis:

Map Contributors:

The first thing I looked at was different contributors to the crowd sourced map data. I queried the database using:

```
SELECT round(count(*) * 100 / sub1.total_entries, 3), user, count(*)
FROM Nodes, (SELECT count(*) AS total_entries FROM Nodes) AS sub1
GROUP BY uid
ORDER BY count(uid) DESC
LIMIT 10;
```

To output the top ten contributors as well as their percent and quantity of contributions:

```
(user,          number of lines          percent)
```

('Mr_Brown', 86600, 29.082)
('Didger85', 65082, 21.856)
('woodpeck_fixbot', 32477, 10.906)
('MikeM44', 22625, 7.598)
('russdeffner', 20824, 6.993)
('tekim', 20420, 6.857)
('joelholdsworth', 13817, 4.64)
('GPS_dr', 13366, 4.489)
('Chris Lawrence', 2653, 0.891)
('CragMapper', 2302, 0.773)

There are 297,777 contributions to this map data. The most contributions goes to the user: "Mr_Brown" with 86,600 contributions accounting for 29.1%.

The top three contributors account for 61.9% of the data.

The top five contributors account for 76.5% of the data.

The top eight contributors account for 92.5% of the data

The total number of contributors are 237 with 229 of those contributing less than 1% of the data.

From this data, it seems likely that the majority of the data is contributed by bots; some of the names, such as ‘woodpeck_fixbot’ and ‘GPS_dr’ support this as well as the sheer number of contributions. Those contributing a small number (<1%) could be a mix of active users and scripts, though it is unsupported.

Eateries

When I attended school in Fort Collins, it was common knowledge that the city boasted the highest ratio of restaurants per capita in the USA. Myth or not, the people there are very proud of restaurants, so I decided to query a breakdown of the restaurant types listed in Open Street Maps:

```
SELECT key, value, count(*) as num
FROM (SELECT key, value FROM Nodes_Tags UNION ALL SELECT key, value FROM
    Ways_Tags) as sub
WHERE key = 'cuisine'
GROUP BY value
ORDER BY num DESC
LIMIT 5;
```

Outputting:

(‘cuisine', 'mexican', 28)
('cuisine', 'american', 24)
('cuisine', 'pizza', 22)
('cuisine', 'burger', 21)
('cuisine', 'sandwich', 17)

With a population of

```
SELECT key, value FROM Nodes_Tags WHERE key = 'population';
```

yielding 148,612 and:

```
SELECT count(*)
```

```
FROM (SELECT key, value FROM Nodes_Tags UNION ALL SELECT key, value FROM
    Ways_Tags) as sub
WHERE key = 'cuisine';
```

yielding 195 listed restaurants.
That is paltry when other cities such as San Francisco boast over 3500 restaurants (Frommer's). However, it is likely that the Open Street Map data does not include every restaurant since it is updated by volunteers.

Conclusion:

Overall the dataset was surprisingly clean. There were a few incongruencies and a few inaccuracies that were detected. The largest confusion I encountered when cleaning the data was the fact that not only was Fort Collins, CO included in the dataset, but also many of the surrounding suburbs and overlapping other cities in close proximity. These were left in place as they were technically correct information. Instead, the expectations were updated, and the data was likewise audited for correctness. If a user desires only Fort Collins, it is preferential to query the database with a specification for postal codes only within desired city, as postal codes map correctly to their designation city.
Postal codes specifically had a few inconsistencies with how they were reported initially (ie "CO 80525" rather than "80525"). This was easily corrected. There was a single incident of an altogether incorrect postal code referencing an otherwise locatable address. This required specific attention via manual research.
Street types seems notoriously inconsistent in these open collection map entry, which required a mapping function to unify them into a consistent pattern, using the full street type (ie "str." becomes "Street"). This section also contained a few errors including dropping street types altogether, spelling errors, and addresses inserted incorrectly into the street column. These were all corrected with with manual and generic mappings to desired format.

Future Suggestions:

A topic this paper has already broached is analyzing restaurants and ratios. Seeing as it appeared there were only a modicum of restaurants included in the open street maps data, it would be an admirable pursuit to build a web scraper to increase the restaurant count to more realistic levels. This is an extremely complicated task, however, and would best be accomplished by searching the web for sites already categorizing city restaurants, and just parsing through their data using a module like BeautifulSoup or an ElementTree if possible. The benefits would be significant, as such a parser could likely fill gaps in other values, however, this program would have to be very comprehensive, and would require catering parameters to each site chosen to parse.

Sources:

http://www.expertmarket.com/USPS-street-suffix

when writing my regular expression to obtain the most common abbreviations for street types

http://effbot.org/zone/element-iterparse.htm#incremental-parsing

for additional reading on iterative parsing.

https://docs.python.org/2/library/sqlite3.html

as a primary resource for sqlite3

https://docs.python.org/2/library/xml.etree.elementtree.html

for elementtree documentation

http://docs.python-cerberus.org/en/stable/install.html

for cerberus documentation

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

as a primary resource for regular expressions

http://stackoverflow.com/questions/21257899/writing-a-csv-file-into-sql-server-database-using-python

for help optimizing my sql import and making it more concise

http://stackoverflow.com/questions/305378/list-of-tables-db-schema-dump-etc-using-the-python-sqlite3-api/33100538#33100538

for sql database analysis

http://data.mongabay.com/igapo/zip_codes/metropolitan-areas/metro-alpha/Fort%20Collins-Loveland%20(CO)1.html

to reference postal codes to their districts

http://www.infoplease.com/us/census/data/colorado/fort-collins/demographic.html

to reference Fort Collins household size

https://www.trulia.com/blog/trends/eating-towns-drinking-towns/

to reference San Francisco restaurant density