P3 - Wrangle OpenStreetMap Data

Student: Peter Carsten Petersen

June 17, 2016

Map Area: Dallas, TX, United States

For this project i have chosen to investigate the Dallas Texas area as found in openstreetmap.org
https://www.openstreetmap.org/export#map=10/32.6914/-96.3913

I have chosen Dallas as this was the first city I visited in USA, and since data nearer to my home in Denmark, did not have same amount of data points as in US data, and fewer obvious areas of data cleaning, as we do not have same issues when dealing with abbreviated street name endings. Also since country is relatively small, there is a unique set of 4-digit postal codes for the entire country, i.e. no use of added state pre-fix which could be in need of clean.

Part1 - Identifying problems in dataset and fix

After initially downloading a small sample size of the Dallas area **(dallas_test.osm)** and running it against an audit.ipynb file **('P3 - Audit file.ipynb')**, I identified three main problems which needed cleaning.

Inconsistent street names ("Road/Rd/Rd. - Drive/Dr.", House numbers included before and after street name)

Inconsistent housenumbers ("7171", "972-788-2591", "1490 W Spring Valley Rd")

Inconsistent postal codes ("TX 75320", "75320")

Inconsistent Street Names

The cleaning of Street names is done with the following code, and tested against data for success.

This code which tested effectively removes all house numbers from Street Name and ensures conformity across the data, i.e. "Rd/Rd./Road" all become Road.

In [9]:
```
## From audit above we can see issues with Dr./Drive and Rd/Rd./Road as well as numbers included in street Names
## which is corrected via the following code.
## Following first run of this code further issues were identified. Street names including "Dallas" / "tx" as well as starting
## with numbers, all which have been included in the code for cleaning.

mapping = { "Dr.": "Drive",'Rd.': 'Road','Rd': 'Road', '#500E': '', 'dallas': '', 'tx': '', 'W': 'West'}


def update_name(file, mapping):
    for event, elem in ET.iterparse(file):
        if (elem.tag == 'tag') and (elem.attrib['k'] == 'addr:street'):
            name = elem.attrib['v']
            n = street_type_re.search(name)
            if n:
                street_type = n.group()

                if street_type.isdigit():
                    m = street_type_re.split(name)
                    name = m[0]


            for word in name.split():
                if word in mapping.keys():
                    name = name.replace(word,mapping[word])
                elif word.isdigit():
                    name = name.replace(word,'')
                    name = name.lstrip(' ')
                    name = name.capitalize()

            return name
```

Housenumbers

Cleaning House numbers included a dual challenge, first of all certain house numbers also included the street name, which was an easy clean with below code. Secondly some house numbers were clearly incorrect, being phone numbers and not house numbers. The latter cleaning is somewhat more difficult and doubtful if can be done with existing data in the openstreetmap record. Possible cleaning could be to match the phone number with phone directory and return the house number, use the Lat/Lon coordinates and retrieve house number from GPS source or simply update manually. However these cleaning options I believe are outside the scope of this assignment, so for this task i have included code which will identify these records and replace with text “To be updated”.

The following code takes care of these cleaning options and was tested against data with success.

In [13]:
```
## From audit above we can see issues with telephone no. instead of house number and house number with address following
## In order to correct this the following code will iterate over the house number and update with "To be updated" if including
## "-" reflecting it is a telephone number and if lenght of data is >5 digits, remove everything after 5th digit.


def update_house(file):
    for event, elem in ET.iterparse(file):
        if (elem.tag == 'tag') and (elem.attrib['k'] == 'addr:housenumber'):
            house_no = elem.attrib['v']
            if '-' in house_no:
                house_no = 'To be updated'

            elif len(house_no) > 5:
                house_no = house_no[:5]

            return house_no
```

Postal Codes

Cleaning postal codes was the easier task, as the only cleaning needed was to remove the possible pre-fix ‘TX’ from data, so that all entries followed a strict 5-digit only format. The following code takes care of this and was tested against test data with success.

In [17]:
```
## From audit above of postcodes we can see issues with a preceeding 'TX'
## In order to correct this the following code will iterate over the postcodes and remove 'TX' when found.


def update_postcode(file):
    for event, elem in ET.iterparse(file):
        if (elem.tag == 'tag') and (elem.attrib['k'] == 'addr:postcode'):
            post_co = elem.attrib['v']
            if 'TX' in post_co:
                post_co = post_co.replace('TX ','')


            return post_co
```

2. Data Overview

For answering following basic statistics and queries in MongoDB, the .osm file for Dallas Texas has been converted to .json format using the submitted code from the Case study on openstreetmap.org (see 'P3 - Audit file.ipynb' for detailed code), and consequently loaded into MongoDB via command prompt code on the MongoDB file folder.

File sizes

dallas_texas.osm ..... 587 MB
dallas_texas.osm.json .... 632 MB

File with all code below included in the github repository 'P3 Mongo Dallas.ipynb'

Number of documents

```
db.dallas.count()
```

Answer: 2.838.472

Number of nodes

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

Answer: 2.551.583

Number of ways

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

Answer: 286.889

Number of unique users

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

Answer: 1.704

Top 1 contributing user

```
top1_user = db.dallas.aggregate([{"$group":{"_id":"$created.user",
"count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":1}])
```

Answer: {u'count': 1120139, u'_id': u'woodpeck_fixbot'}

Number of users appearing only once (having 1 post)

```
only_once = db.dallas.aggregate([{"$group":{"_id":"$created.user",
"count":{"$sum":1}}}, {"$group":{"_id":"$count",
"num_users":{"$sum":1}}}, {"$sort":{"_id":1}}, {"$limit":1}])
```

Answer: {u'num_users': 319, u'_id': 1}
 # "_id" represents postcount

3. Additional queries and improvement idea

Contributor statistics

Top user contribution percentage ("woodpeck_fixbot") - 43.90%
Combined top 2 users' contribution ("woodpeck_fixbot" and "fmmute") - 47,79%
Combined Top 10 users contribution - 66.49%
Combined number of users making up only 1% of posts - 1312 (about 77% of all users)

Top 10 appearing amenities

```
top_amen = db.dallas.aggregate([{"$match":{"amenity":{"$exists":1}}},
{"$group":{"_id":"$amenity","count":{"$sum":1}}},
{"$sort":{"count":-1}}, {"$limit":10}])
```

Answer:
{u'count': 3802, u'_id': u'parking'}
{u'count': 2859, u'_id': u'place_of_worship'}
{u'count': 2003, u'_id': u'school'}
{u'count': 1065, u'_id': u'fast_food'}
{u'count': 958, u'_id': u'restaurant'}
{u'count': 452, u'_id': u'fuel'}
{u'count': 246, u'_id': u'grave_yard'}
{u'count': 218, u'_id': u'bank'}

{u'count': 165, u'_id': u'fire_station'}
{u'count': 155, u'_id': u'post_box'}

Biggest religion

```
religion = db.dallas.aggregate([{"$match":{"amenity":{"$exists":1},
"amenity":"place_of_worship"}},{"$group":{"_id":"$religion",
"count":{"$sum":1}}},{"$sort":{"count":-1}}, {"$limit":1}])
```

Answer: {u'count': 2769, u'_id': u'christian'}

Most popular cuisines

```
cuisine = db.dallas.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"fast_food"}},
{"$group":{"_id":"$cuisine",
"count":{"$sum":1}}},{"$sort":{"count":-1}}, {"$limit":1}])
```

Answer: {u'count': 372, u'_id': u'burger'}

“None” entries and improvement suggestion

For both **Religion** and **Cuisine** the second highest grouping is classified as **None:**

Religion: {u'count': 72, u'_id': None}
Cuisine: {u'count': 323, u'_id': None}

In order to improve this lack in data, i suggest that a webscraping technique could be employed on websites such as **www.tripadvisor.com** and **www.yellowpages.com,** where the address data is matched against the entry in one of the repositories and cuisine/religion is returned and updated in openstreetmap data.

This technique if done correctly could be an easy fix for the missing data, but there are a number of aspects which should be considered beforehand:

1. The amount of data missing and need of cleaning should be of a significant size and of enough importance for the task/question to be answered, before time and resource is spend on cleaning. As examples the question above for “Biggest Religion” would not yield a different answer if the remaining 72 missing data points were cleaned, i.e. do not clean unless needed otherwise. For “Most popular Cuisine” however the cleaning of missing 323 data points, could theoretically change the answer, i.e. must be cleaned for answer to be fully verified.

2. A webscrabing solution for data cleaning is only going to be as good as the data quality on the target database, so before initiated we must be sure that the quality of data we are “scraping” is sufficient for our needs, and post this exercise we must audit the new data.

3. Before we initiate a clean, we should audit the missing data points, in order to review if there is a possible pattern, i.e. same user, same area etc. If pattern found there could be a systematic error which could be corrected, rather than employing new cleaning procedure.

Sources:

Sample projects provided in course material.
Wiki on the openstreetmap.org data