OpenStreetMap Data Case Study

Map Area

```
Map: Chicago, IL, USA

Map Selection dimensions: minlat="41.8307000" minlon="-87.7142000"
maxlat="41.9590000" maxlon="-87.5926000"

The data included in this document is from www.openstreetmap.org. The data is
made available under ODbL.

Interpreter: conda:base (Python 3.7.7)
```

The map selection is based on Chicago, the largest city in Illinois. OSM Data will be downloaded in XML format, with data quality being audited, locating potentially problematic data, performing data cleanup and exporting the cleaned up data to a database.

Map data Autiting

Included libraries

In [1]:
```
import xml.etree.cElementTree as ET
import pprint
import re
import csv
import codecs
import cerberus
import schema
import sqlite3
import pandas as pd
from collections import defaultdict
from mapparser import *
from audit import *
from tags import *
```

Importing the OSM file for investigation.

In [2]:
```
osm_file = r'Chicago.xml'
```

In this task I will take a quick look at the OSM data to figure out what tags it contains and its counts to get a general idea about the data.

In [3]:
```
tags = count_tags(osm_file)
pprint.pprint(tags)
```

```
defaultdict(<class 'int'>,
            {'bounds': 1,
             'member': 94865,
             'meta': 1,
             'nd': 1330459,
             'node': 1083005,
             'note': 1,
             'osm': 1,
             'relation': 2483,
             'tag': 1061586,
             'way': 179501})
```

I will look at "k" values for each tag to determine if there any potential problems by using 3 regular expression: "lower", for tags that contain only lowercase letters and are valid, "lower_colon", for otherwise valid tags with a colon in their names, "problemchars", for tags with problematic characters and "other" for other tags that do not fall into the other three categories.

In [4]:
```
pprint.pprint(process_map(osm_file))
```

```
{'lower': 346602, 'lower_colon': 441379, 'other': 273605, 'problemchars': 0}
```

Problems encountered in your map

Auditing Street Names

While looking at a sample of the data, I've noticed that some street names and orientation seems to be over-abbreviated. I would prefer to see data unabbreviated for clarity.

In [5]:
```
street_types = audit(osm_file)[0]
```

Based on the findings that several street names/direction is abbreviated, the following abbreviated to unabbreviated map will be used.

Which results in the following conversion

In [6]:
```
    for street_type, ways in street_types.items():
       for name in ways:
        better_name = update_name(name, mapping)
        print (name, "=>", better_name  )
```

```
North Lake Shore Drive West => North Lake Shore Drive West
South Riverside Plaza => South Riverside Plaza
W Merchandise Mart Plaza => West Merchandise Mart Plaza
Merchandise Mart Plaza => Merchandise Mart Plaza
North Riverside Plaza => North Riverside Plaza
West Wolf Point Plaza => West Wolf Point Plaza
East Riverwalk South => East Riverwalk South
W. Riverwalk South => West Riverwalk South
West Fulton Market => West Fulton Market
W Irving Park Rd => West Irving Park Road
W. Madison St. => West Madison Street
North Michigan Ave => North Michigan Avenue
West Chicago Ave => West Chicago Avenue
W Jackson Blvd => West Jackson Boulevard
Chicago Riverwalk => Chicago Riverwalk
North Sangamon => North Sangamon
West Churchill Row => West Churchill Row
North River Walk => North River Walk
North Breakwater Access => North Breakwater Access
North May Streets => North May Street
```

Auditing Zip Codes

In [7]:
```
postal_code_types = audit(osm_file)[1]
```

In [8]:
```
pprint.pprint(dict(postal_code_types))
```

```
{'60064': {'60064'},
 '60067': {'60067'},
 '60622-4580': {'60622-4580'},
 '606476': {'606476'}}
```

Suprisingly only few issues encounted. Let's keep date consisatant with 5 digit zip code for standardization.

Zip Code in error: 60064 2110 North Seminary Avenue Correct zip code: 60614

Zip Code in error: 60067 108 North Green Street Correct zip code: 60607

Zip Code in error: 60622-4580 (not really an error, it has suffix in different zip code scheme. But let's standardize it to 5 digit scheme) 2657-2659 West Walton Street Correct zip code: 60622

Zip Code in error: 606476 1750 North Milwaukee Avenue (Which is a bar. I bet whoever entered the zip code is a frequent there. (just kidding)) Correct zip code: 60647

After the initial audit above, the data will be cleaned up by using functions in audit.py by converting over-abbreviated names into non-abbreviated ones & correcting zip code to adjusted one. OSM XML file will be parsed into tabular .csv format to prepare data to be inserted into SQL database.

Note: modified UnicodeDictWriter for Unicode/UTF-8 handing and Python 3 compatability.

Note: modified process_map function for Unicode/UTF-8 handing and Python 3 compatability.

Data Overview (SQL):

```
File sizes:

Chicago.xml: 278,729 KB
OpenStreetMapChicagoRev2b.db: 158,672 KB
nodes.csv: 95,252 KB
nodes_tags.csv: 3,954 KB
ways.csv: 11,536 KB
ways_nodes.csv: 31,310 KB
ways_tags.csv: 36,858 KB
```

In [9]:
```
connection = sqlite3.connect(r'OpenStreetMapChicagoRev2b.db')
```

Number of Nodes

In [10]:
```
sqlquery = """
SELECT COUNT(*) FROM nodes;
"""
df = pd.read_sql_query(sqlquery, connection)
df
```

Out[10]:

	COUNT(*)
0	1083005

Number of Ways

In [11]:
```
sqlquery = """
SELECT COUNT(*) FROM ways;
"""
df = pd.read_sql_query(sqlquery, connection)
df
```

Out[11]:

	COUNT(*)
0	179501

Number of Unique users

In [12]:
```
sqlquery = """
SELECT COUNT(DISTINCT(UID))AS "Number of Unique Users"
FROM (SELECT UID FROM NODES UNION ALL SELECT UID FROM WAYS);
"""
df = pd.read_sql_query(sqlquery, connection)
df
```

Out[12]:

	Number of Unique Users
0	1141

Top 10 Users

In [13]:
```
sqlquery = """
SELECT E.USER as "User Name", COUNT(*) AS "Updates"
FROM (SELECT USER FROM NODES UNION ALL SELECT USER FROM WAYS) E
GROUP BY E.USER
ORDER BY 2 DESC
LIMIT 10
"""
df = pd.read_sql_query(sqlquery, connection)
df
```

Out[13]:

	User Name	Updates
0	chicago-buildings	932949
1	nickvet419	72247
2	Umbugbene	31755
3	jimjoe45	24421
4	Zol87	22492
5	Chicago Park District GIS	20943
6	Steven Vance	20286
7	bbmiller	13004
8	Rallysta74	12472
9	NE2	11413

Top 10 popular amenities

In [14]:
```
sqlquery = """
SELECT VALUE AS "Amenity Name", COUNT(*) AS  "Count"
FROM NODES_TAGS
WHERE UPPER(KEY)="AMENITY"
GROUP BY 1
ORDER BY 2 DESC
LIMIT 10;
"""
df = pd.read_sql_query(sqlquery, connection)
df
```

Out[14]:

	Amenity Name	Count
0	restaurant	1396
1	bar	566
2	cafe	459
3	bicycle_rental	371
4	fast_food	317
5	bench	273
6	place_of_worship	226
7	school	152
8	bank	91
9	drinking_water	73

In [15]:
```
sqlquery = """
SELECT value AS "Cuisine Types", COUNT(*) as Count
FROM nodes_tags
JOIN (SELECT DISTINCT id FROM nodes_tags WHERE value="restaurant") nodes_ids
ON nodes_tags.id=nodes_ids.id
WHERE UPPER(KEY)="CUISINE"
GROUP BY 1
ORDER BY COUNT(*) DESC
LIMIT 10;
"""
df = pd.read_sql_query(sqlquery, connection)
df
```

Out[15]:

	Cuisine Types	Count
0	american	102
1	mexican	63
2	pizza	49
3	italian	42
4	chinese	32
5	thai	20
6	japanese	15
7	sandwich	14
8	indian	13
9	sushi	12

Summary / ideas about the dataset

While auditing the data, to my surprise data I looked at was in pretty decent shape, other than minor potential issues. One of them is address/street naming convention and some zip codes are incorrect, some of the street names were over-abbreviated and some zip codes were incorrect which brings unnecessary potential complications in data processing.

My suggestion is to create a parser or a script, that would automatically convert/translate between over-abbreviated and non-abbreviated naming patterns and a criteria for valid zip codes. There is not a given accepted standard in un/abbreviation convention and people tend to use both. While it's usually not an issue to a casual user, programs that handle the data seem to prefer standardized data (or whatever we tell them to).

The main challenge of creating parser script is being able to anticipate the data that users enter than validate it through subset of rules and to facilitate language translation between different regions. It’s an open source platform that includes variety of languages. Each country/region uses several ways that address/location information is being recorded. It would require massive collaboration of people from different countries, cultures, regions to agree upon the convention data should be recorded than formatting it according to several data formats. As mentioned earlier some people can be keen to record the data in one way or another and coming up with single convention would be very tedious task.

One way to address it is to continuously run variety of audits to analyze the most common issues encountered and adjust validation rules accordingly. Another one is to cross reference data users enter on OpenStreetMap with address/location data that Postal Offices/Shipping companies use all over the globe, since they are able to handle to handle large amount of languages, schemes, formats validating with their data might be of great use.

```
References:
    * Case Study OpenSteetMap Data [SQL] by udacity
    * https://stackoverflow.com/questions/5838605/python-dictwriter-writing-utf-8-encoded-csv-files
      (UTF-8/Unicode conversion)
```