Data wrangling on open street map data

Map Area

Ontario City, Canada

- https://download.geofabrik.de/north-america/canada/ontario.html

I first tried to choose my hometown or my current city but I didn't get big enough data, so after some thought I randomly selected one city and downloaded the data.

Problems encountered in the map

- The first problem was the size of data, it was too small and too big.
- Street name
- Tags attribs in data
- Tag values with colon
- Processing time

Data Size:
At first the size of data was too small like I already mentioned in the previous session. Then after I choose the city Ontario, I downloaded the data and it was over 13GB, which was now too big. So I again had to get data which was 1GB.

Street names:
The problem in this case was not abbreviations but the number of types of street. I had to run the code multiple times to filter the proper names of street. Every time I was getting a big list of names of streets that I missed.

Tags attribs:
When the data was too big I decided to download the sample data for the same city which was some 15MB, but I downloaded it from different mirror site. After I made my code work for sample file, I found out that the number of tag attributes in sample file and main data file are different. Then I created a sample file from main data and worked on it.
<node changeset="0" id="694341568" lat="43.5813313" lon="-79.6185508" timestamp="2016-05-05T18:29:47Z" version="6">
For example this tag doesn't include user or uid attribute.

Tag values:
The tags with colon values took some time. I had to work on the code to parse them properly.

Processing time:
After all this when I was ready to run code on my main data. It was too big and took many hours to parse.
Overall the data was almost clean there were not many problems.

Auditing Data

For the auditing purpose I first looked at the sample to find the street names.

```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)


expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square",
"Lane", "Road", "North", "South",
            "Trail", "Parkway", "Commons", "Line", "Way", "East", "West", "Sideroad",
"Crescent", "Circle", "Private"]

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)
```

I used the above code for this. The expected list is the list of all the valid streets in the area. Every name that is not in this list is filtered in street_types.

Every time this returned a long list of street names, I had to take valid names from the results, add to the expected list and run the code again.

After I find the list with invalid names I created a dict for the mappings for all the invalid names and used that to clean data using the below mentioned code

For example Hurontario St to Hurontario Street

And Eastbourne Crescent, I didn't know about Crescent but there were many streets with that type so I added this to expected list.

But there were still few exceptions like "Townline Road Middleton North Walsingham" and Highways with numbers like "Highway 35"

```
mapping = { "St": "Street",
            "St.": "Street",
            "Ave": "Avenue",
            "Ave.": "Avenue",
            "Blvd": "Boulevard",
            "Pl": "Place",
            "Dr": "Drive",
            "Ct": "Court",
            "Sq": "Square",
            "Sq.": "Square",
            "Rd": "Road",
            "Rd.": "Road",
            "Pky": "Parkway",
            "Pkwy": "Parkway"
            }
```

```
def update_name(name, mapping):
    for n in name.split():
        if n in mapping:
            name=name.replace(n,mapping[n])
            break
    return name
```

Data overview

This section contains basic statistics about the data.

File sizes

```
ontario.osm ........... 1 GB
osm.db ................ 553 MB
nodes.csv ............. 373 MB
nodes_tags.csv ........ 25.2 MB
ways.csv .............. 11.3 MB
ways_tags.csv ......... 33 MB
ways_nodes.cv ......... 175 MB
```

Number of nodes

```
  sqlite> SELECT COUNT(*) FROM nodes;
```

6584542

Number of ways

```
  sqlite> SELECT COUNT(*) FROM ways;
```

323261

Distinct postcodes in area

```
  sqlite> SELECT COUNT(DISTINCT(value)) FROM NODES_TAGS WHERE key='postcode';
```

12

So we have a total of 6584542 nodes in the area and 6584542 ways. Also we have 12 distinct postcodes in the area.

Distinct house numbers in area

```
sqlite> SELECT COUNT(DISTINCT(value)) FROM NODES_TAGS WHERE key='postcode';
```

14182

And we have 14182 houses in the area.

Additional exploration

Top ten appearing amenities

```
sqlite> SELECT COUNT(*) AS num
        FROM NODES_TAGS
        WHERE key='amenity'
        GRAOUP BY value
        ORDER BY num DESC
        LIMIT 10;
```

```
post_box              632
restaurant            441
fast_food             394
bench                 310
place_of_worship      229
café                  184
parking               143
fuel                  132
waste_basket          129
bicycle_parking       114
```

Biggest religion

```
sqlite> SELECT nodes_tags.value, COUNT(*) AS num
        FROM nodes_tags
          JOIN (SELECT DISTINCT(id) FROM nodes_tags
                      WHERE value='place_of_worship') i
          ON nodes_tags.id=i.id
        WHERE nodes_tags.key='religion'
        GROUP BY nodes_tags.value
        ORDER BY num DESC
        LIMIT 1;
```

```
christian     97
```

Conclusion

The full data for this is very big over 13GB, This is the analysis of only part of full data. The same code can be used on the full data to explore more.

This data had few problems, this means there was less manual updates, most of the data was auto updated using GPS and apps. This data file did not include user data, which can be interesting feature to explore, to find the kind of user fill the data, which area are they more active etc.

Other Ideas

This was a really great and almost clean data, but in my opinion the data needs some specific instructions to be added to make this more consistent. For example with the street name we need to define either it can be full name or abbreviation, some names can be singular or plurals.

Also an additional validation check can be added. For example the data entered by a user needs to get validated by either other users or by some other API.

This may also represent some issues. We can’t give suggestions for every scenario, different places have different ways of naming places or roads. In India we have mains and cross roads in a few places. In America we have West road, North roads, and then highways with numbers etc. We can’t have a universal set of instructions for the users.

For another scenario, if we ask other users to validate places, some places might never get validated. This will limit the availability of the data and the use of another api will get very to work with.