OSM Project Report

September 24, 2017

1 OpenStreetMap Data Wrangling Project

1.1 Map Area

I live in **Istanbul**, so detecting unusual elements and standardize key attributes is easier than other cities for me. On the other hand, it is challenging due to Turkish characters and different formats especially for street names and phone numbers. Findings from this project may be affect my daily life so I choose this city.

Map link of Istanbul,Turkey: OpenStreetMap

Data is dowloaded from Mapzen by selecting OSM XML.

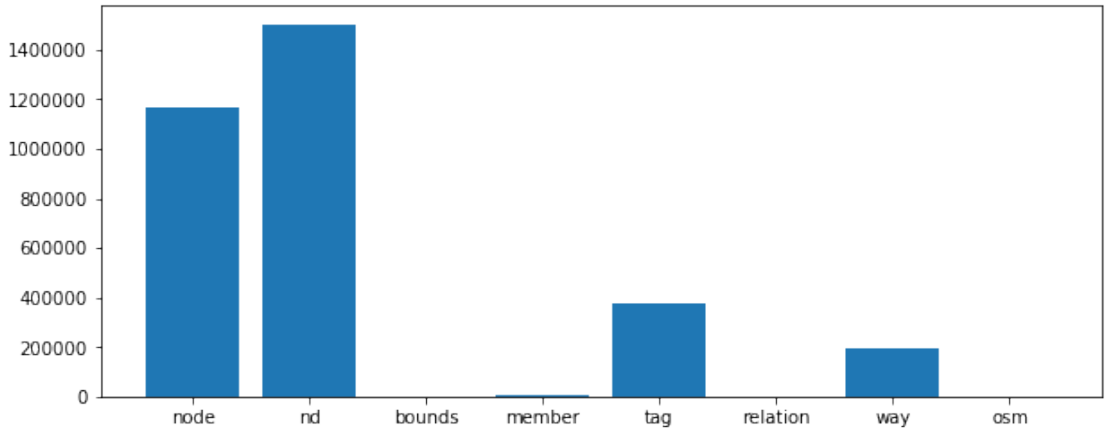
1.2 Auditing the Data

Count the number of unique element types:

```
In [7]: pprint.pprint(tags) #See appendix for codes
        fig
```

```
{'bounds': 1,
 'member': 8140,
 'nd': 1502262,
 'node': 1168391,
 'osm': 1,
 'relation': 696,
 'tag': 375872,
 'way': 192994}
```

```
Out[7]:
```



1.2.1 Types of Tag

Before processing the data and adding it into your database, we should check the "k" value for each tags and see if there are any potential problems.

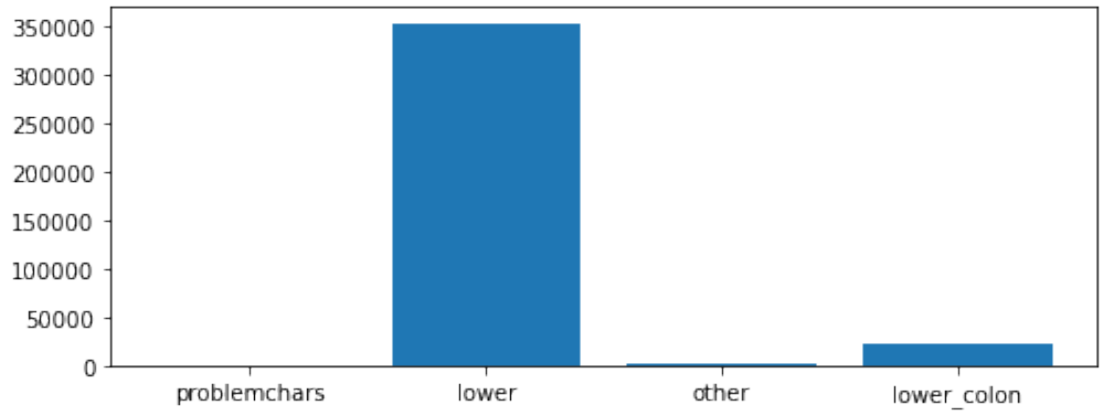
3 regular expressions are provided to check for certain patterns in the tags. we would like to change the data model and expand the "addr:street" type of keys to a dictionary like this: {"address": {"street": "Some value"}}

We have a count of each of four tag categories in a dictionary with the function 'key_type': * "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 [8]: pprint.pprint(keys) #See appendix for codes
        fig1
```

```
{'lower': 352646, 'lower_colon': 22219, 'other': 1002, 'problemchars': 5}
```

Out[8]:



Sample from OSM XML data:

```
<node changeset="44355605" id="4551323037" lat="40.9597535" lon="29.091451" timestamp="2016-12-
<tag k="name" v="Simay Antik"/>
<tag k="shop" v="antiques"/>
<tag k="addr:city" v="stanbul"/>
<tag k="addr:street" v="Turucu Dere Sokak"/>
<tag k="addr:housenumber" v="25"/>
</node>
```

1.3 Problems Encountered in Map

First we start with taking each one of the 50 elements and build a sample file as referenced at project page by running *sample.py* .

After that, we created audit.py file and make investigation possible problems in data.

1.3.1 Street Names

1. **Abbreviations**: Sk. -> Sokak | Sk -> Sokak | Cd. -> Caddesi | Cad. -> Caddesi | Bulv. -> Bulvar |
2. **Lowercase/Uppercase**: sokak -> Sokak
3. ** Paragoge(Additional letters)**: soka -> Sokak

These are handled problems. Other unsolved problems that neighboorhood names,suburb names,shopping mall names, apartment numbers or full adresses were written by people as street name.

1.3.2 Inconsistent Phone Numbers

Actually, phone numbers are consistent in Turkey and also in Istanbul. - **+90**: the country code, - **212**: city code for European side of Istanbul or **216**: city code for Anatolian side of Istanbul - **7-digits**: Remained part of phone number

However, data was entered in many different way.

- 0212 123 45 67
- +90 212 123 45 67

- +90 2121234567
- +90 212 1234567
- 0 212 1234567
- +90 212 123 4567
- 212 123 4567

Our target was removing all spaces between digits and add country code when it needs to reach **+9021201234567** or **+9021601234567** format.

1.4 Data Cleaning

After the audit data, we created data.py to clean programatically some inconsistency in street names and phone numbers.

1.4.1 Street Names

With help of the regex and mapping dictionary we created, we search and made corrections programmatically by this update_name function in audit.py. We call this function in data.py for tags of node and way.

```
In [ ]: def update_name(name, mapping):
            m = street_type_re.search(name)
            if m.group() in mapping.keys():
                name = re.sub(m.group(), mapping[m.group()], name)
            return name
```

1.4.2 Phone Numbers

With help of the regex, we standardized phone numbers by using update_phone function in audit.py.Same as street names, we also call this function in data.py for tags of node and way.

```
In [ ]: def update_phone(phone):
            if re.compile(r'^\d{4}\s\d{3}\s\d{2}\s\d{2}$').search(phone):
                phone = phone.replace(' ', '')
                # 0212 123 45 67 -> +9021201234567
                return  '+9'+phone #
            elif re.compile(r'^\+90\s\d{3}\s\d{3}\s\d{2}\s\d{2}$').search(phone):
               # +90 212 123 45 67 -> +9021201234567
                return phone.replace(' ', '')
            # is continued..
```

Also, we convert XML file by using iterparse to iteratively step through each top level element in the XML,shaping each element into several data structures,utilizing a schema and writing each data structure to these csv files seperately: - nodes.csv - nodes_tags.csv - ways.csv - ways_nodes.csv - ways_tags.csv

1.5 Overview of the data with SQL

We parse the elements in the OSM XML file, transforming them from document format to tabular format, thus making it possible to write to .csv files. These csv files can then easily be imported to a SQL database as tables.

Database.py is used to create SQL database by using csv files.

File Size

- istanbul_turkey.osm -> 243 MB
- mydb.db -> 177 MB
- nodes.csv -> 92,7 MB

- nodes_tags.csv -> 2,51 MB
- ways_nodes.csv -> 35,7 MB
- ways_tags.csv -> 10,6 MB
- ways.csv -> 11,1 MB

```
In [33]: import pandas as pd
         import sqlite3
         conn = sqlite3.connect("mydb.db")
         cur = conn.cursor()
```

Unique Users Number

```
In [34]: query= "SELECT COUNT(DISTINCT(e.uid))\
                     FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;"
         df = pd.read_sql_query(query, conn)
         df

Out[34]:    COUNT(DISTINCT(e.uid))
         0                    2348
```

Number of Ways

```
In [35]: query= "SELECT COUNT(*) FROM nodes"
         df = pd.read_sql_query(query, conn)
         df

Out[35]:    COUNT(*)
         0   1168391
```

Number of Nodes

```
In [36]: query= "SELECT COUNT(*) FROM ways;"
         df = pd.read_sql_query(query, conn)
         df

Out[36]:    COUNT(*)
         0    192994
```

Most contributing 5 users

```
In [37]: query="SELECT e.user, COUNT(*) as num\
                     FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e\
                     GROUP BY e.user \
                     ORDER BY num \
                     DESC LIMIT 5;"

         df = pd.read_sql_query(query, conn)
         df
```

```
Out[37]:           user     num
         0        Nesim  101875
         1  bigalxyz123   86279
         2     Cicerone   63318
         3    Ckurdoglu   49323
         4    katpatuka   49072
```

Most popular 5 suburbs

```
In [38]: query= "SELECT value, COUNT(*) as num\
                     FROM nodes_tags\
                     WHERE key='suburb'\
                     GROUP BY value\
                     ORDER BY num DESC LIMIT 5;"
         df = pd.read_sql_query(query, conn)
         df
```

```
Out[38]:         value  num
         0       Fatih  104
         1     Beyolu   46
         2    Ümraniye   35
         3     Üsküdar   33
         4  Arnavutköy   32
```

Most common 5 amenities

```
In [39]: query= "SELECT value, COUNT(*) as num\
                     FROM nodes_tags WHERE key='amenity'\
                     GROUP BY value \
                     ORDER BY num DESC LIMIT 5;"
         df = pd.read_sql_query(query, conn)
         df
```

```
Out[39]:         value   num
         0    pharmacy  2440
         1  restaurant   860
         2        cafe   739
         3        bank   493
         4        fuel   335
```

Most popular 5 cuisines

```
In [40]: query = "SELECT nodes_tags.value, COUNT(*) as num\
                     FROM nodes_tags \
                     JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') \
                     i ON nodes_tags.id=i.id WHERE nodes_tags.key='cuisine' \
                     GROUP BY nodes_tags.value \
                     ORDER BY num DESC LIMIT 5;"
         df = pd.read_sql_query(query , conn)
         df
```

```
Out[40]:       value  num
          0   turkish   96
          1     kebab   28
          2  regional   26
          3     pizza   12
          4   seafood   10
```

1.5.1 Other ideas about the dataset

People should enter data in more formal way. There should be some templates for specific information like phone numbers for country.On the other side, this is open project to everyone and if there will be strict rules; participation may decrease.

To richen the data set, people can rate places. According to this ratings, we can analyze best places in city. In addition, we can analyze most popular places in city by adapting check-in feature to OSM.

1.5.2 Resources

- Regex : https://regex101.com
- Udacity Discussion Forum

1.5.3 Appendix

These appendix includes codes used for figures and calculations at the beginning of the report.

```
In [ ]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
        import xml.etree.cElementTree as ET
        import pprint
        import matplotlib.pyplot as plt
        filename ="istanbul_turkey.osm"
        tags = {}
        for event, elem in ET.iterparse(filename):
            if elem.tag in tags:
                tags[elem.tag] += 1
            else:
                tags[elem.tag] = 1
        pprint.pprint(tags)

        T =tags
        fig = plt.gcf()
        plt.bar(range(len(T)), T.values())
        plt.xticks(range(len(T)), T.keys())
        fig.set_size_inches(10, 4, forward=True)
        cc=plt.show()
```

```
In [ ]: import re
        lower = re.compile(r'^([a-z]|_)*$')
        lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
```

```
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')
def key_type(element, keys):
    if element.tag == "tag":
        if problemchars.search(element.attrib['k']):
            keys['problemchars'] += 1
        elif lower.search(element.attrib['k']):
            keys['lower'] += 1
        elif lower_colon.search(element.attrib['k']):
            keys['lower_colon'] += 1
        else:
            keys['other'] += 1
    return keys

def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)
    return keys

keys = process_map(filename)
pprint.pprint(keys)

K =keys
import matplotlib.pyplot as plt
fig1 = plt.gcf()
plt.bar(range(len(K)), K.values())
plt.xticks(range(len(K)), K.keys())
fig1.set_size_inches(8, 3, forward=True)
plt.show()
```