Project 3 – Wrangle OpenStreetMap

Christian Guzman

June 2017

Map Area
New York, New York, United States

• New York City Map Area
• Source: https://mapzen.com/data/metro-extracts/

• The map area I chose contains parts of the Bronx, Manhattan, Queens, and Brooklyn. Relatively speaking my map has a higher representation of Bronx Manhattan and Queens than Brooklyn in my extraction. I wanted to focus on the tri-borough area more specifically. I am interested in what insights and problems in the dataset can be found through database querying. The map also contains a small portion of area in New Jersey due the method of extraction.

Problems Encountered

• Abbreviated street types

The street names in the data appeared to be inconsistently abbreviated. Some street names ended in “Ave”, “Pl” while others contained the full spelling “Avenue” and “Place” for example.

• There can only be one city

Even though the data is mainly derived within the borders of New York City there appeared to be non-cities like neighborhoods being labeled as cities. In this data we should see most if not all cities to be “New York”.

Abbreviated Street Types

In auditing for street names there were inconsistent abbreviations, there were street names that ended in ‘Blvd‘ or ‘Boulevard’, ‘Pl’ instead of ‘Place’. To remedy the following regular expressions were used to locate the street types.

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

The above expression can be used within a function to check whether or not the street type is incorrectly abbreviated. However, there needs to be a second regular expression to account for street names with abbreviation “Sr”(State Road) which usually appears before the street type.

```
sr_re = re.compile(r'\bSr\b')
```

```
def update_name(name, mapping):
    '''Locates and updates abbreviated street types
    for each street name.'''
    m = street_type_re.search(name) #street type match object
    state_road = sr_re.search(name) #state road match object
    if m and m.group() in mapping.keys():
        name = street_type_re.sub(mapping[m.group()], name)
    if state_road:
        name = sr_re.sub(mapping[state_road.group()], name)
    return name
```

Using the above, street names like “Horace Harding Expressway Sr S” become Horace “Harding State Road South”

There Can Only Be One City

While auditing the data for city names I expected to find borough names such as Bronx, Brooklyn and Queens but there were also many ‘cities’ that were really just neighborhoods within the boroughs. Specifically many were names of Queens neighborhoods.

```
{'Astoria': 20,
 'Bronx': 1,
 'Brooklyn': 34,
 'Corona': 6,
 'Edgewater': 4,
 'Elmhurst': 3,
 'Flushing': 4,
 'Forest Hills': 4,
 'Jamaica': 18,
 'Kew Gardens': 1,
 'Long Island City': 8,
 'New York': 193,
 'New York City': 2,
 'New York city': 1,
 'Queens': 5,
 'Rego Park': 3,
 'Roosevelt Island': 1,
 'Sunnyside': 1,
 'Woodside': 3}
```

There’s also the distinction of naming New York itself as there are a few elements within the document have city name ‘New York City’ as opposed to just ‘New York’. To correct this and improve uniformity of city names I used the following function so that the main city name is ‘New York’.

```
city_mapping = ['new york city', 'brooklyn', 'bronx', 'queens', 'staten island',
                'astoria', 'corona', 'elmhurst', 'flushing', 'forest hills', 'jamaica',
                'kew gardens', 'long island city', 'rego park', 'roosevelt island',
                'sunnyside', 'woodside']

def update_city_name(name, mapping):
    if name.lower() in mapping:
        name = 'New York'
    return name
```

Rather than simply substituting ‘New York’ for all names in city I chose to only substitute for names that were found in the audit since the map extract does include a small portion of New Jersey.
Snippet of output showing what code does:

```
Queens => New York
Jamaica => New York
Brooklyn => New York
Astoria => New York
Flushing => New York
Elmhurst => New York
```

Sorting city, state, and county by count, descending

Using the ‘countQuery’ function below I wanted to see if the input data for these keys were following a uniform format.

```
def countQuery(key, limit=10):
    '''
    Sorts the values of 'key' argument by count descending
    number of results limited by value of 'limit' argument.
    '''
    cursor.execute('''SELECT tags.value, COUNT(*) AS total FROM
                        (SELECT * FROM nodes_tags
                        UNION ALL
                        SELECT * FROM ways_tags) AS tags
                        WHERE tags.key = '{}'
                        GROUP BY tags.value
                        ORDER BY total DESC
                        LIMIT {};'''.format(key, limit))
    results = cursor.fetchall()
    return results
```

```
countQuery('city', limit=100)
```

```
[(u'New York', 2963),
 (u'Edgewater', 32),
 (u'Jackson Heights', 11),
 (u'Fresh Meadows', 6),
 (u'Glendale', 6),
 (u'Middle Village', 6),
 (u'Whitestone', 5),
 (u'Brooklyn, NY', 4),
 (u'Maspeth', 4),
 (u'New York, NY', 4),
 (u'East Elmhurst', 3),
 (u'Ridgewood', 3),
 (u'BRONX, NY', 2),
 (u'College Point', 2),
 (u'Bayside', 1),
 (u'Blissville', 1),
 (u'Bowery Bay, NY', 1),
 (u'Brooklyn, New York', 1),
 (u'East Elmhurst, NY 11370', 1),
 (u'Flushing, NY', 1),
 (u'Jackson Heights, NY 11370', 1),
 (u'M', 1),
 (u'New York NY', 1),
 (u'Queens, NY', 1),
 (u'Sunnyside, NY', 1),
 (u'Woodside (Queens)', 1),
 (u'new york', 1)]
```

While the majority of the tags have city name: “New York” as we should expect there are still Queens neighborhoods that are being put in as ‘city’. Other errors include borough names “Bronx, NY” and “Brooklyn, New York” and other inconsistencies like a single lowercase “new york” or “New York, NY” instead of the proper “New York”. There is also the ‘M’ character input as a city name. The only ‘non-new york’ city name is “Edgewater” which is in New Jersey.

There are also inconsistencies for state in county.

```
countQuery('state', limit=100)
```

```
[(u'NY', 2102), (u'NJ', 32), (u'New York', 14), (u'ny', 14)]
```

```
countQuery('county', limit=100)
```

```
[(u'Queens, NY', 6986),
 (u'Bronx, NY', 2520),
 (u'New York, NY', 2000),
 (u'Kings, NY', 666),
 (u'Bergen, NJ', 85),
 (u'Queens, NY; Kings, NY', 38),
 (u'Kings, NY; Queens, NY', 31),
 (u'Kings, NY;Queens, NY', 12),
 (u'Kings, NY:Queens, NY', 11),
 (u'Queens, NY;New York, NY', 9),
 (u'New York, NY; Kings, NY', 8),
 (u'h', 6),
 (u'Bronx, NY:New York, NY', 3),
 (u'Bronx, NY; Queens, NY', 3),
 (u'Queens, NY; New York, NY', 3),
 (u'New York, NY;Kings, NY', 2),
 (u'Queens, NY; Bronx, NY; Queens, NY', 1),
 (u'Queens, NY;Kings, NY', 1),
 (u'\x7fQueens, NY', 1)]
```

In truth for locations in New York City we should see the borough names as counties.

Very small issue with postcodes

```
query = cursor.execute('''SELECT tags.value, COUNT(*) AS total FROM
                        (SELECT * FROM nodes_tags
                        UNION ALL
                        SELECT * FROM ways_tags) AS tags
                        WHERE tags.key = 'postcode' GROUP BY tags.value
                        HAVING total < 4
                        ORDER BY total DESC;''')
printQuery(query, all=True)
```

```
[(u'10055', 3),
 (u'10152', 3),
 (u'10153', 3),
 (u'10154', 3),
 (u'11207', 2),
 (u'83', 2),
 (u'10001', 1),
 (u'10003', 1),
```

As you can see here there is a postcode with only 2 digits. But of the 244,676 tags with postcodes(not including the tiger postcodes) ‘83’ was the only erroneous postcode which I found to be impressive.

Looking at the code I wrote to ensure uniformity of postcodes, I can improve the cleaning by taking into account codes smaller than 5-digits. My code did ensure that postcodes were only made up of numbers and no special or space characters. In the sample audit all postcodes were 5 characters in length.

Overview Statistics of Data

File sizes

```
newyork.osm..............................: 449MB
nodes.csv................................: 161MB
nodes_tags.csv...........................: 6MB
nymetro.db...............................: 313MB
sample.osm...............................: 45MB
ways.csv.................................: 21MB
ways_nodes.csv...........................: 57MB
ways_tags.csv............................: 55MB
```

Number of nodes

```
def printQuery(query, all=False):
    query
    if all:
        pprint.pprint(cursor.fetchall())
    else:
        print cursor.fetchall()[0][0]

query = cursor.execute('''SELECT COUNT(*) FROM nodes;''')
printQuery(query)
```

```
1714872
```

Number of ways

```
query = cursor.execute('''SELECT COUNT(*) FROM ways;''')
printQuery(query)
```

```
311461
```

Number of unique users

```
query = cursor.execute('''SELECT COUNT(DISTINCT(uniques.uid)) FROM (SELECT uid FROM nodes
                  UNION
                  SELECT uid FROM ways) AS uniques;
                    ''')
printQuery(query)
```

```
1338
```

Top 10 contributing users

```
query = cursor.execute('''SELECT elems.user, COUNT(*) as total
                          FROM (SELECT user FROM nodes
                          UNION ALL
                          SELECT user FROM ways) elems
                          GROUP BY elems.user
                          ORDER BY total DESC
                          LIMIT 10;''')

printQuery(query, all=True)
```

```
[(u'Rub21_nycbuildings', 1386338),
 (u'ingalls_nycbuildings', 323934),
 (u'Korzun', 32696),
 (u'ingalls', 24126),
 (u'celosia_nycbuildings', 23342),
 (u'aaron_nycbuildings', 22134),
 (u'mikercpc', 19668),
 (u'robgeb', 17479),
 (u'MCM7', 12040),
 (u'woodpeck_fixbot', 9940)]
```

Contributor statistics

```
sqlite> SELECT COUNT(*)
   ...> FROM
   ...>     (SELECT e.user, COUNT(*) as num
   ...>      FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
   ...>      GROUP BY e.user
   ...>      HAVING num=1)  u;
425
```

425 of users only had 1 post. The contributions is top-heavy with user: “Rub21_nycbuildings” comprising 68% of the contributions, and the user with the second most contributions being “ingalls_nycbuildings” with approximately 16% of the contributions.

Additional Ideas for Analysis

While exploring the data I thought it would be interesting for future analyses to involve comparing the boroughs and the densities of different types of nodes. Living in New York you

may believe that some boroughs are better serviced in some regards than others. While the entirety of the Bronx only has 2 movie theaters, Manhattan has many more and even two in one street! It would be interesting to see the differences between the boroughs in this manner.

Potential problems
A problem with this analysis is the nature of OpenStreetMap and its user input. Tags describing nodes, ways, and areas are mostly optional so many of the differences could be caused by some users filling out more details than others. There's also the fact that some areas see more activity than others, and may be more complete. Despite this, useful insights could still be drawn.

Additional Data Exploration

Top 10 amenities

```
countQuery(key='amenity', limit=10)
```

```
[(u'bicycle_parking', 1289),
 (u'restaurant', 782),
 (u'place_of_worship', 674),
 (u'parking', 627),
 (u'school', 525),
 (u'cafe', 273),
 (u'fast_food', 212),
 (u'embassy', 191),
 (u'drinking_water', 159),
 (u'bank', 148)]
```

Places of leisure

```
countQuery(key='leisure', limit=10)
```

```
[(u'pitch', 1066),
 (u'park', 753),
 (u'playground', 231),
 (u'garden', 189),
 (u'sports_centre', 47),
 (u'picnic_table', 43),
 (u'swimming_pool', 33),
 (u'stadium', 24),
 (u'track', 23),
 (u'dog_park', 13)]
```

Types of shops

```
countQuery(key='shop', limit=10)
```

```
[(u'supermarket', 165),
 (u'convenience', 141),
 (u'clothes', 87),
 (u'hairdresser', 56),
 (u'yes', 56),
 (u'deli', 53),
 (u'bakery', 49),
 (u'laundry', 43),
 (u'alcohol', 40),
 (u'beauty', 29)]
```

Top 10 sports

```
countQuery(key='sport', limit=10)
```

```
[(u'basketball', 298),
 (u'tennis', 289),
 (u'baseball', 229),
 (u'handball', 83),
 (u'soccer', 71),
 (u'team_handball', 26),
 (u'volleyball', 16),
 (u'swimming', 11),
 (u'running', 8),
 (u'american_football', 7)]
```

Number of delis

```
query = cursor.execute('''
                       SELECT COUNT(*) as total FROM (SELECT * FROM nodes_tags
                       UNION ALL SELECT * FROM ways_tags) AS t
                       WHERE t.value LIKE '%deli';
                       ''')
printQuery(query)
```

```
112
```

Top 10 cuisines

```
printQuery(cursor.execute('''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 10;'''), all=True)
```

```
[(u'italian', 38),
 (u'pizza', 35),
 (u'american', 34),
 (u'mexican', 30),
 (u'chinese', 21),
 (u'indian', 16),
 (u'thai', 14),
 (u'burger', 12),
 (u'japanese', 11),
 (u'french', 10)]
```

Conclusion

While the data was very large, it was not as messy as I first expected. Data with large amounts of user input will likely have errors. Besides formatting, much of these errors can come from differences in what is perceived to be correct information. Such with the case of city names, New York is unique in that it is the only US city with prominent boroughs.

In general while filling out forms of any kind the borough is often used as the city name and not New York for example. In terms of mailing addresses Queens has its own rules. Instead of listing “Queens, NY” it is customary to write the name of the neighborhood, like “Ridgewood, NY”. All this, I imagine, contributes to the confusion.

To keep with consistency, I would relegate the boroughs to county, and keep “New York” as the sole city, which I believe is how most would prefer since the majority of ‘city’ names are labeled as “New York”. Through continuation of the python script with mapping out all of the mislabeled values and replacing them with “New York.” can the data be further cleaned.