Data Wrangling Project

In this project I used data wrangling techniques, such as assessing data for validity, accuracy, completeness, consistency and uniformity to clean OpenStreetMap data for a part of the world that I care.

Map Area

Cupertino, California, USA

Why I chose this area?

I chose Cupertino, CA, USA because this is where I currently live. It is my home and am very familiar with this city. Cupertino is a U.S. city in Santa Clara County, California. The city is renowned as a center of innovation in Silicon Valley. Cupertino is famous as the home of high-tech giant Apple Inc. Within the 13 square miles, this city has a population of 64,000 people. This city is known for its excellent public schools. More than 60% of its residents aged 25 years and older hold a bachelor's degree or higher, and more than 40% were born outside of the United States. Cupertino is 42 miles south of San Francisco. As part of this data wrangling project, I wanted to check the quality of the Cupertino map data from OpenStreetMap.org.

Cupertino data was exported from https://www.openstreet.org and I downloaded a XML OSM dataset. My dataset is about 55.3 MB (uncompressed). I used Overpass API and downloaded a custom square area.

https://www.openstreetmap.org/relation/2221709

Steps taken on this project

1. Acquire and generate a sample data
2. Count the types of tags available in the data
3. Audit the data for potential problems
4. Clean the data by fixing the problems
5. Prepare the data to be inserted into a SQL database

The steps above served as a good guide for me as it helped me get through the data wrangling project carefully. It helped me to successfully go through the auditing and cleaning process of this dataset. Eventually, transferring the cleaned data to .csv files.

Issues on the Map

For this Data Wrangling project, I decided to audit the street names and the postal codes. The purpose of this audit is to identify any potential problems. The dataset was audited to see if there are any issues with the street names and postal codes. I noticed a few problems:

1. Abbreviated street names: N Blaney Ave. and Stevens Creek Blvd.
2. A combination of 5 digit postal codes and 9 digit postal codes
3. Wrong postal code format: CA 95014
4. Incorrect postal code : CUPERTINO

i. Abbreviated Street Names

One of the obvious errors is the street names. For example, "Boulevard" may occasionally be written as "Blvd".

From the audit street names, I encountered problems with two streets. These two street names required correction. They were "N Blaney Ave" and "Stevens Creek Blvd". I needed to correct "Ave" and "Blvd" to "Avenue" and "Boulevard" respectively.

A mapping dictionary is created to change "Ave" and "Blvd" to "Avenue" and "Boulevard". By using the mapping dictionary, we create an 'update_name()' function to clean the street name. The 'update_name' function converts the old street name with the new street name. If the street name has the defined string in the mapping dictionary then the change will happen.

```
# Dictionary of street types that I decided to reformat by using "mapping"
mapping = { "Ave" : "Avenue",
            "Blvd": "Boulevard"
            }

def update_name(name, mapping):
    for key,value in mapping.iteritems():
        if key in name:
            return name.replace(key,value)
    return name
```

After running the code, the problematic street names were fixed. For example, Stevens Creek Blvd was corrected to Stevens Creek Boulevard and N Blaney Ave was fixed to N Blaney Avenue.

ii. Inconsistent Postal Codes

The next audit and cleaning process was the postal codes. From the results of the postal codes, I encountered several problems. Majority of the postal codes were either in 5 digit postal codes or 9 digit postal codes. In addition, there is a "CA" in one of the postal code and we need to fix this problem. Also, one of the postal codes says "CUPERTINO" instead of the postal code digits. We need to clean these problems.

I decided to standardize the postal codes to 5 digits in the cleaning process. Similar to the 'update_name' function in the street auditing, the 'update_postcode' function fixes all the postal codes by standardizing to 5 digits, removing the 'CA' and any other postal codes that is different from the 5 digit postal code display.

```
def update_postcode(name):
    if "-" in name:
        name = name.split("-")[0].strip()
    elif "CA" in name:
        name = name.split("CA ")[1].strip('CA ')
    elif len(str(name))>5:
        name=name[0:5]
    elif name.isdigit()==False:
         name=00000
    return name
```

Count the types of tags available in the data

We are using the iterative parsing to process the map file. We are going to find out not only what tags are there but also how many tags. This is to get a sense on how much of which data we can expect to have in the map. The count_tags function should return a dictionary with the tag name as the key and the number of times this tag can be encountered in the map as value.

```
def count_tags(filename):

    tags = {}
    for event, elem in ET.iterparse(filename):
        if elem.tag in tags.keys():
            tags[elem.tag] += 1
        else:
            tags[elem.tag] = 1

    return tags
def test():

    tags = count_tags('cupertino.osm')
    pprint.pprint(tags)


if __name__ == "__main__":
    test()
```

```
{'bounds': 1,
 'member': 8970,
 'meta': 1,
 'nd': 291147,
 'node': 260421,
 'note': 1,
 'osm': 1,
 'relation': 402,
 'tag': 117807,
 'way': 27227}
```

From the results above, we know that there are 10 different types of tags within this data. The most common tags within this data are 'nd', 'node', 'tag' and 'way'. We will primarily focus on these four common tags in our auditing and cleaning process.

Prepare the data to be inserted for SQL Database

After auditing and cleaning is complete, the next step is to prepare the data to be inserted into a SQL database. I used sqlite3 to create the database. We will 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. Moreover, in the conversion process, the data was validated against a predefined schema.py file to ensure that both the structure of the csv files and the types of the data entered were as expected.

Data Overview

Here are the summary of the files:

OSM file ……. 55.3 MB

nodes.csv …….. 21.6 MB

nodes_tags.csv …… 476 KB

ways.csv ……. 1.73 MB

ways_tags.csv …….. 3.43 MB

ways_nodes.csv …… 6.98 MB

Number of Nodes

```
query = "SELECT count(DISTINCT(id)) FROM nodes;"
c.execute(query)
rows=c.fetchall()

pprint.pprint(rows)
```

```
[(260421,)]
```

Number of Ways

```
query = "SELECT count(DISTINCT(id)) FROM ways;"
c.execute(query)
rows=c.fetchall()

pprint.pprint(rows)
```

```
[(27227,)]
```

Number of Unique Users

```
query = "SELECT COUNT(DISTINCT(e.uid))FROM (SELECT uid FROM Nodes UNION ALL SELECT uid FROM Ways) as e;"
c.execute(query)
rows=c.fetchall()

pprint.pprint(rows)
```

```
[(354,)]
```

Top Ten Contributing Users

```
query = "select e.user, count(*) as num from (select user from nodes UNION ALL select user from ways) as e group by user order by num desc limit 10;"
c.execute(query)
rows=c.fetchall()
print 'Top 10 contributing users and their contribution:\n'

pprint.pprint(rows)
```

```
Top 10 contributing users and their contribution:

[(u'dannykath', 37269),
(u'n76_cupertino_import', 34989),
(u'karitotp', 31529),
(u'calfarome', 27247),
(u'RichRico', 24668),
(u'andygol', 23536),
(u'samely', 15590),
(u'Luis36995', 13993),
(u'Eureka gold', 10852),
(u'doug_sfba', 7136)]
```

Top 10 Amenities

```
query = "SELECT value, COUNT(*) as num FROM Nodes_tags WHERE key = 'amenity' GROUP BY value ORDER BY num desc limit 10;"
c.execute(query)
rows=c.fetchall()
print 'Top 10 amenities:\n'

pprint.pprint(rows)
```

```
Top 10 amenities:


[(u'bench', 123),
(u'restaurant', 69),
(u'parking_entrance', 36),
(u'bicycle_parking', 34),
(u'cafe', 25),
(u'toilets', 19),
(u'fast_food', 14),
(u'bank', 11),
(u'fuel', 7),
(u'place_of_worship', 7)]
```

Biggest Religion

```
query = "SELECT Nodes_tags.value, COUNT(*) as num FROM Nodes_tags JOIN (SELECT DISTINCT(id) FROM Nodes_tags WHERE value = 'place_of_worship') as sub ON Nodes_tags.id = sub.id WHERE Nodes_tags.key = 'religion' GROUP BY Nodes_tags.value ORDER BY num DESC LIMIT 1;"
c.execute(query)
rows=c.fetchall()
print 'Biggest Religion:\n'

pprint.pprint(rows)
```

```
Biggest Religion:

[(u'christian', 7)]
```

Most Popular Food

```
query = "SELECT Nodes_tags.value, COUNT(*) as num FROM Nodes_tags JOIN (SELECT DISTINCT(id) FROM Nodes_tags WHERE value = 'restaurant') as sub ON Nodes_tags.id = sub.id WHERE Nodes_tags.key = 'cuisine' GROUP BY Nodes_tags.value ORDER BY num DESC LIMIT 15;"
c.execute(query)
rows=c.fetchall()
print 'Most Popular Food:\n'

pprint.pprint(rows)
```

```
Most Popular Food:

[(u'chinese', 9),
(u'pizza', 6),
(u'japanese', 5),
(u'asian', 3),
(u'thai', 3),
(u'ice_cream', 2),
(u'sandwich', 2),
(u'burger', 1),
(u'greek', 1),
(u'indian', 1),
(u'international', 1),
(u'italian', 1),
(u'korean', 1),
(u'mexican', 1),
(u'noodle', 1)]
```

Ideas for Improvement

From the query about the "Most Popular Food", I noticed that the Indian food shows only "1". However, there are more than one Indian food in the Cupertino area. Next, "Asian" food is listed as one of the most popular food. In my opinion, Asian food could be Chinese, Indian, Thai, Japanese, etc. Hence, the question of accuracy, completeness and uniformity arises on the data quality.

How can we encourage more users to participate in the OSM data? How frequently should the users update the OSM data to reflect the latest information on the map? How do we educate the users on the field units so it conforms to a standard format? One thing that might address these problems is to provide incentives to the users so that they are motivated to enter the data accurately to the OSM map in a timely manner.

The quality of the the openstreetmap is directly dependent on the users. When there are more user participation in the openstreetmap then it is also prone to human errors. These errors impact the map data quality. Perhaps we can have a system in place to address the validity, accuracy, completeness, consistency and uniformity of the data. In addition, we can have bots fill map information in rural areas.

Conclusion

Auditing and cleaning the data is a long process. It is by far the most crucial part in data analysis. Data scientists can spend up to 50 - 80 % of their time auditing and cleaning data. Data wrangling is an iterative procedure. Just as I had expected, I had to work through these auditing and cleaning steps several times so that the data can be easily explored and analyzed. Unfortunately, it is difficult to identify mistakes and where these mistakes are embedded in the data. Hence, going through the auditing and cleaning process several times help identify these errors. As a result, we can make accurate decisions which can impact business results.