Jostin Flake

OpenStreetMap Data Case Study

Map Area

Honolulu, Hawaii, United States

- [https://www.openstreetmap.org/relation/119231 (https://www.openstreetmap.org/relation/119231))

I chose this city on the island of Hawaii because it is my absolute favor place to go on vacation. I mean, who doesn't love Hawaii? I am very excited to use the data in the "Honolulu.osm" data to discover new and interesting facts about this awesome city!

Creating a sample Map from our Map

The first task performed after exporting our full sized Map from "OpenStreetMap" was to create a sample sized Map for auditing purposes. This allows us to audit a smaller portion of the data set and saves us time from having to run our scripts against the full sized Map. Auditing is an important process inside of Data Wrangling as it allows us to investigate the data. Getting to know the data can be a powerful tool as it will often lead to the discovery of problematic data. The python script that was used to perform the task of creating our sample map is titled **gatherSample.py**

Becoming familiar with the data using sample Map

The following section outlines will provide the names and purposes for the some of the python scrips that were used to become more familiar with the dataset. All of these scripts were ran against the sample Map to save time and expedite the initial phase of auditing. Scripts such as these allow us to explore the data and will help us to become more familiar with its content.

- **exploring_users.py - returns the number of unique users in the sample Map.**
- **count_tags.py - returns the tag name and amount of times that tag is found in the sample Map.**

It is important to become familiar with the data as this step will generally be the stepping stone that is needed in order to better determine what direction you will take in the "Data Auditing" phase of Data Wrangling.

Problems Encountered in the Map

Data Auditing

In this section of the project we will discuss the some of the problems that were encounted in the Map.

Abbreviated Street Names

The following abbreviations are street names that we found in the "Honolulu.osm" Map that did not match our expected names for streets. The values generated by the **audit_ways.py** and **audit_nodes.py** scripts which would return any values from specific tag attributes with the key of "addr:street". We cross checked the "expected" variable which contained a list of acceptable names(e.g., Street, Drive, or Road) and if the value provided did not match we added the name to a dictionary that would store only unique values.

- **Ave**
- **Blvd**
- **Blvd.**
- **Dr**
- **Hwy.**
- **St**
- **street**
- **Pl**
- **Rd**

Tags from Tiger GPS pulled in address data and stored the information in multi-leveled tags.

Below is an example of a street address stored in multiple levels of Tiger GPS tags. Notice the keys "tiger:name_base" and "tiger:name_type" make up the street name.

```
<tag k="tiger:county" v="Honolulu, HI" />
<tag k="tiger:name_base" v="Wilder" />
<tag k="tiger:name_type" v="Ave" />
```

City names that were spelled incorrectly

There were also a bunch of city names that were spelled incorrectly in the Map. After the database was built, the following query was ran to provide the names for all the cities in the Map and the amount of times those names appeared.

```
SELECT tags.value, COUNT(*) as count_cities
FROM (SELECT * FROM nodes_tags
      UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key='city'
GROUP BY tags.value
ORDER BY count_cities DESC;
```

Results

City	Count_cities
Honolulu	1346
Aiea	22
Kailua	13
Kaneohe	11
Pearl City	8
Honlulu	5
Fort Shafter	4
Waipahu	4
Waimanalo	3
Ewa Beach	2
Honollulu	1
Kailuna	1
honolulu	1
waimanalo	1

Fixing Street Names in Tags

Data Cleaning

Now that some problem areas have been identified lets look at a way to go about correcting the problems. The section below will outline the steps taken to correct the inconsistent naming schema found for street names within the tags for Nodes and Ways.

The information provided through auditing the data allowed us to progam a way to correct these incosistencies by using "regular expressions". The python script **data.py** was responsible for iterating each tag in the element tree. If the tagtype was equal to "node" or "way" it would place specific attributes for their child tags into a dictionary. During this process if one of the child tags had the attribute of "addr:street" it would take the last word in the string value for that tag, and check to see if that word matched a key inside of a "mapping" dictionary. If the word such as "st" was found, it would then replace that word with the "regular expression" provided as the value for "st". For example "street" would become "Street" with a capital "S", and "Rd." would be changed to "Road". After cleaning up the data, and placing it into a dictionary object, the data for these objects were then written to csv files titled "nodes.csv", "nodes_tags.csv", "ways.csv", "ways_nodes.csv", and "ways_tags.csv".

```
#regular expressions to be searched for in our "update_name" function
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

#function to update street names
def update_name(name, mapping):


    m = street_type_re.search(name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            name = re.sub(street_type_re, mapping[street_type], name)

    return name

#cleaning function for street names
if i.attrib['k'] == 'addr:street':
    node_dict['value'] = update_name(i.attrib['v'], mapping)
```

The code above is some of the most integral pieces that work effectively inside of our **data.py** script to update the street name. Of course the code isn't whole in this representation but i felt it was important to highlight these critical functions. To provide evidence that the script in "data.py" worked I have first ran the original "audit_ways.py" script above which will return the same problematic street names it did during the auditing process.

Results from running "verify_update_names.py"

Incorrect Street names that still exist:

No incorrect street names exist!

Using sqlite we took our csv's that were exported from the "data.py" script and created a database containing a table for each individual csv. This enables the ability to run a query inside of the python script above to retreive all the values for for streets an store them in a variable called rows. The script then continues to do a crosscheck on all of the problematic abbreviations found in the "audit_ways" script and if the value still exists to print that value. If no values are found to exist we return the results shown above "No incorrect street names exist!". This provides evidence that our cleaning method inside of "data.py" worked. The code that performs the producure that was just explained reside in a script titled **verify_update_names.py**

Fixing Misspelled City Names in Tags

Data Cleaning

As mentioned in the "problem area" of this document. After the original script **data.py** had executed successfully. Its common to then create a database using the .csv data so that SQL queries can provide quick and powerful analysis of the data. More on this process will be discussed in the next section, however, it is common to find problems with the data after reaching that point, and this is why Data Wrangling is an iterative process of auditing and cleaning many times. In the "problems area" we covered what SQL query was used to discover the cities with incorrect names.

```
#regular expressions to be searched for in our "update_city" function
city_type_re = re.compile(r'\w+.?\w+', re.IGNORECASE)

#function to update city names
def update_city(name, city_mapping):

    m = city_type_re.search(name)
    if m:
        city_type = m.group()
        if city_type not in expected_cities:
            name = re.sub(city_type_re, city_mapping[city_type], name)

    return name

#cleaning city name function
elif i.attrib['k'] == 'addr:city':
    node_dict['value'] = update_city(i.attrib['v'], city_mapping)
```

In order to correct the mispelled city names a very similar approach used to fix our street names was also used for fixing the city names in tags. Slight modifications to the regular expression variable that was being used to search had to be made as there were city's with names that contained two words seperated by spaces inside of the value string. Above highlights the most critical functions in resolving this specific problem in the data. This code was added to the orignal **data.py** script and then that script was executed again.

Some Statistical Analysis on the Data

In this section we will use SQL queries from our newly created database to provide some statistical analysis on the data. Additional information will be provided about possible other problem areas.The challenges and benefits that will come from addressing those issues will also be discussed.

File Sizes

Below is a list of file name and the size of the associated file. These files were all used for the project

Honolulu.osm ..... 55.9 MB

openstreetmap_honolulu.db .... 30.8 MB

nodes.csv ..... 21.2 MB

nodes_tags.csv ..... 0.59 MB

ways.csv ..... 1.75 MB

ways_tags.csv ..... 3.19 MB

ways_nodes.cv ..... 7.34 MB

Number of Nodes

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

result: 257840

Number of Ways

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

result: 29976

Number of Unique Users

```
SELECT COUNT(DISTINCT(all_uids.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) all_uids;
```

result: 753

Number of Shops in Nodes

```
SELECT COUNT(*) FROM nodes_tags WHERE key = 'shop';
```

result: 319

Most popular Cuisine to Eat

Below is the results from a query that shows the most cuisine to eat in Honolulu.

```
SELECT value, COUNT(*) FROM nodes_tags WHERE key = 'cuisine' GROUP BY key ORDER BY
COUNT(*) DESC LIMIT 1;
```

result: Japanese.

Additional insights

One of the more problematic issues I noticed with this dataset was how different sources would format the same type of data in different ways. For example, the Tiger GPS would bring in multi-leveled tags for different segments in a street. Here is an SQL query I ran to return the value, key, and number of times that specific key was found in the "ways_tags" table of our database, the results are shown below the query.

```
SELECT value, key, COUNT(*) AS count_tiger_tags
FROM ways_tags
WHERE type='tiger'
GROUP BY key
ORDER BY count_tiger_tags DESC;
```

Results

value	key	count_tiger_tags
Honolulu, HI	county	4252
A41	cfcc	4211
Mokunoio	name_base	3882
PI	name_type	3634
no	reviewed	3406
Moanalua	name_base_1	119
Rd	name_type_1	110
tiger_import_dch_v0.6_20070809	source	98
203252242	tlid	98
bulk_upload.pl-fd7bfe10-7a28-44e7-b196-d2b35d899d00	upload_uuid	97
S	name_direction_prefix	92
no	separated	83
N	name_direction*prefix*.	11
96706	zip_right	8
State Highway 99	name_base_2	6

To be more accurate, if this was truly some data that was going to be used by a company, we would want to syncronize the format of data. So that all of the same type of data was found in the exact same format accross the dataset. Simply put, prior to iterating over each element in our **data.py** script, we would need to have built additional audit scripts to audit the tiger tags, then build some other cleaning scripts that would take tag types from tiger tags such as "name_type" and "name_type1" and concatenate the values found in those attributes with the values for the beginning of the address strings found for the values in "name_base" and "name_base1". This concatenated string would then be placed in a new string under the way tag as an "addr:street" tag type, if that type of tag was not already found. If it was found, we could do one of two things ignore it the tiger info and delete it from the tags, or check the the existing "addr:street" value and if it didnt match, build another scenario on how to tackle the new problem of two different street addresses belong to the same way tags. As you can see this would be a very complex and complicated problem to tackle so I did not try to accomplish it here. However, its always good to think about innovative ways to resolve data errors.

Conclusion

In conclusion, as demonstrated in this project, data wrangling is an effective and efficient means whereby data analysts can extract, audit, and clean data in preparation for analyzing the data. The methods and functions learned in this course are very valuable as they can save large amounts of times when data wrangling is needed for large sets of data. As the digital world continues to grow and the amounts of data grows along with it, more effective and experienced data wranglers will become increasingly more and more in demand.