Project 3: Wrangle OpenStreetMap Data

Greater Seattle Region

Student: Whitney King

This project will wrangle OpenStreetMap data pertaining to the Greater Seattle region in Washington State, USA. Seattle was chosen, as it's my home city, and it will be interesting to work with data that contains locations I'm personally familiar with.

The dataset was obtained from a preselected region of OSM data hosted by MapZen: https://mapzen.com/data/metro-extracts/metro/seattle_washington/

Objective

The main objective of this project will be to ensure that pertinent dirty data has been cleaned up and corrected prior to converting the data to JSON. Data wrangling will be done in MongoDB using Python3. The bulk of the work was done using Jupyter Notebook.

OSM Dataset

Since the dataset for the Greater Seattle Region is extremely large (1.6 GB), we'll first generate a sample of the data set to assist with preliminary development and analysis. By getting a sample of the data that is less than 100MB, we'll still have a good chunk of the data from the region, but processing time will be significantly less. For the purposes of this stage of the data wrangling, this sample set will work just fine.

For the purpose of this report, only the most important parts of the scripts will be shown

Creating a Sample

The sample script that created the sample used for this audit (as well as the smaller sample included with this project) is self-contained in a Python script, and located here:

https://github.com/WhitneyOnTheWeb/Seattle_OSM_MongoDB/blob/master/CreateSample.py

Running this script will output the sample OSM file name, as well as the size of the file

```
(py3) C:\JupyterNotebook\MongoDB>python.exe C:\JupyterNotebook\MongoDB\CreateSample.py
seattle_small_sample.osm created:
File size:  1.8 MB
```

Assessing Dataset

The assessment of the data to understand the shape was done as part of the initial case study, and the counts of attributes, buildings, and shops can be seen here:

https://github.com/WhitneyOnTheWeb/Seattle_OSM_MongoDB/blob/master/OSMWrangle_GreatSeattle.ipynb

After we've imported the OSM XML data, we can see how the elements are broken down. Counting the tags in the dataset allows us to gain an understanding of the size and structure of the data we're working with.

It's interesting to see the counts of occurrences for each type of element tag in the XML (particularly node and way, as they contain the most interesting fields), however these counts only show a top-level list of elements, not the tags/values nested therein.

Problems Encountered

- The investigation done on this dataset is based on limited understanding of the overall structure behind the OpenStreetMaps data.
- To avoid making assumptions about the meaning or connections between obscure information, this audit chose to focus on cleanup of data with obvious meanings, such as address types.
- Many tags are inconsistent and appear to be subjectively entered by users, so getting them uniform would require a large amount of investigatory work.
- Data points show up under more than one tag, so it could be possible for some counts to have duplicate entries.

Generating Cleaned JSON File / Creating DB

The Python script used to shape the data, create a JSON file, and upload the data into MongoDB is located here:

https://github.com/WhitneyOnTheWeb/Seattle_OSM_MongoDB/blob/master/CleanData.py

Running this script will shape the data, output a preview of the JSON documents, validate tags, clean the data to match the specifications of the mapping values, load the data into MongoDB, and finally execute MongoDB queries that accompany the next section.

```
(py3) C:\JupyterNotebook\MongoDB>python.exe C:\JupyterNotebook\MongoDB\CleanData.py
seattle_small_sample.json created:
File size:  8.4 MB




Sample Shape:
{'address': {'city': None,
             'housenumber': None,
             'postcode': None,
             'state': None,
             'street': None},
 'amenity': None,
 'building': None,
 'created': {'changeset': '214775',
             'id': '25840363',
```

Clean Up Efforts

- Mapped desirable values for street types, and ensured street names matched those values before conversion to JSON
- Corrected ill formatted zip codes, ensuring Canadian zips were in the proper format, as well as Washington
- Added frequently occurring attributes to the shape of the data
    - Many attributes that have interesting pieces of information are very sparsely populated, so any attributes that didn't have a key/value pair for a certain node were populated with `None`.
- Regexes were run to validate tag format, as well as cleanup to street names to match the desired format specified in the mapping dictionary
    - Invalid tags were ignored
    - Tags with colons were parted out into their own organized dictionary by type of tag (`addr` and `tiger`)

Preview of Data Clean Up

```
Corrected Zip Codes:
V9B1V7 => V9B 1V7
v8r 5e9 => V8R 5E9
Olympia, 98501 => 98501
V9B1L8 => V9B 1L8
v8Z 1H1 => V8Z 1H1
V8Z6E4 => V8Z 6E4
V8Z6E6 => V8Z 6E6
V8S2J8 => V8S 2J8
V8T4K7 => V8T 4K7


Corrected Street Names:
Total Ways:  61
Carnation-Duvall Rd NE => Carnation-Duvall Rd Northeast
234th Place NE => 234th Place Northeast
Bellevue Way NE => Bellevue Way Northeast
236th Ave NE => 236th Ave Northeast
University Way NE => University Way Northeast
156th Pl NE => 156th Pl Northeast
180th Pl NE => 180th Pl Northeast
127th PL NE => 127th PL Northeast
Sand Point Way NE => Sand Point Way Northeast
155th Place NE => 155th Place Northeast
```

Additional Clean Up Suggestions

- Timestamp could be matched to separate date/time fields
- TIGER date information is more segmented, and could be used to make a larger address validation system
- Descriptive attributes could be better grouped to have a more informative and consistent dataset

Exploration in MongoDB

All of the exploration that was done in the initial phases is made much simpler once the data is cleaned up and imported into MongoDB. Additionally, MongoDB will enable us to take a much more in depth look at the information with a lot less code. We'll look at some of the previously scraped data points to see how they compare when queried via pymongo now that we've normalized the shape of the most interesting pieces of data. These scripts can be accessed via the CleanData.py script, or the accompanying Jupyter Notebook.

File Size

```
#Original XML Sample File Size
print('XML File: ', str(os.path.getsize(SAMPLE_NAME)/1024/1024), 'Mb')

#JSON File Size
print('JSON File: ', str(os.path.getsize(json_file)/1024/1024), 'Mb')
```

```
XML File:  55.321757316589355 Mb
JSON File:  246.0293664932251 Mb
```

Count Nodes/Ways

```
print('Number of nodes:', collection.find({"type":"node"}).count())
print('Number of ways:  ', collection.find({"type":"way"}).count())
print('Total entries:   ', collection.count())
```

```
Number of nodes: 1030656
Number of ways:   102036
Total entries:   1132692
```

Unique Contributors

```
# Number of unique users
print('Unique Contributors: ', len(db.Sample.distinct("created.uid")))
```

```
Unique Contributors:  1677
```

Count Non-Null Attributes

```
fields = ['id',
     'name',
     'amenity',
     'building',
     'shop',
     'phone',
     'pos']

for field in fields: #Iterates through basic field values to count non-null occurances of each
    print(field + ':', db.Sample.find({field:{"$ne": None}}).count())
```

```
id: 1132692
name: 29400
amenity: 5492
building: 43672
shop: 1452
phone: 484
pos: 1030656
```

Additional Exploration

The simple scripts above give an interesting look at the data now that it's in MongoDB, but it doesn't start to answer a lot of interesting questions that could be asked about the data. Using more in depth pipeline queries into the sample data will reveal more information.

Types of Buildings

```
pl = [{"$group":{"_id": "$building",
                 "count": {"$sum": 1}}},
            {"$sort": {"count": -1}}]
result = list(collection.aggregate(pl))
print('Counts of Building Types: ')
pprint.pprint(result)
```

```
Counts of Building Types:
[{'_id': None, 'count': 1089020},
 {'_id': 'yes', 'count': 38008},
 {'_id': 'house', 'count': 2448},
 {'_id': 'residential', 'count': 1476},
 {'_id': 'apartments', 'count': 368},
```

It seems like most users use the building attribute as a Boolean yes/no value, while others use it as a description of the type of building. This type of inconsistent record keeping could really throw off understanding of how this field should be tracked.

Types of Shops

```
pl = [{"$group":{"_id": "$shop",
                 "count": {"$sum": 1}}},
            {"$sort": {"count": -1}}]
result = list(collection.aggregate(pl))
print('Counts of Shop Types: ')
pprint.pprint(result)
```

```
Counts of Shop Types:
[{'_id': None, 'count': 1131240},
 {'_id': 'convenience', 'count': 212},
 {'_id': 'car_repair', 'count': 116},
 {'_id': 'hairdresser', 'count': 84},
 {'_id': 'beauty', 'count': 80},
 {'_id': 'clothes', 'count': 80}
```

Again, we see 'yes' values sprinkled in with otherwise categorical data. This really drives home the observations of inconsistency in categorical attributes.

Popular Cuisines

```
pl = [{"$match": {"amenity":"restaurant",
                  "cuisine": {"$ne":None}}},
            {"$group":{"_id":"$cuisine",
                       "count":{"$sum":1}}},
            {"$sort":{"count":-1}},
            {"$limit":10}]
result = list(collection.aggregate(pl))
print('Most Popular Types of Food: ')
pprint.pprint(result)
```

```
Most Popular Types of Food:
[{'_id': 'pizza', 'count': 48},
 {'_id': 'mexican', 'count': 44},
 {'_id': 'chinese', 'count': 24},
 {'_id': 'japanese', 'count': 20},
 {'_id': 'asian', 'count': 20},
 {'_id': 'italian', 'count': 16},
 {'_id': 'burger', 'count': 16},
 {'_id': 'american', 'count': 16},
 {'_id': 'indian', 'count': 16},
 {'_id': 'thai', 'count': 12}]
```

For a fun last exploration, we'll look at the most popular cuisines in Seattle. This is based on data we know to be inconsistent, so even within this dataset, we can assume this is not a complete list of restaurants, or the food they serve. However, it's at least able to give us an idea of what is popular in the area. Mmm, pizza!

Conclusion

After auditing this data, it's clear that having such a wide contributor base into such an open-ended system creates to a lot of dirty data. There is a lot of room for improvement not only in how information is tracked, but also in how it's associated. Many of the things that would need attention to really clean this information up would benefit from running automated jobs and getting creative with regexes. That can be difficult, expensive, and time consuming.

On the plus side, there is a dedicated user base, and a wide range of information. If descriptive data points such as attributes were better tracked, there would be an amazing amount of useful information that could be dug out of this dataset. That said, it's still interesting and informative data, given it's understood to be entered very subjectively.

References

- Udacity Discussion Forums
- Udacity: Data Wrangling with MongoDB
- http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-file-generated-via-xml-etree-elementtree-in-python
- http://stackoverflow.com/questions/2104080/how-to-check-file-size-in-python