Open Street Map for the City of Melbourne

Introduction

In this project, we have downloaded data for the city of Melbourne using `MapZen`.The original data is a XML file. As a part of Data Wrangling, the goal is to clean the xml file and transform it into a JSON file before loading into MongoDB database. In the following sections, we perform following actions; cleaning the XML file, transforming into JSON format, loading into MongoDB, and querying the database to find some interesting information.

Sample Data Overview

Here are some basic information regarding the input file, `sample_file.osm`;

XML File size : 172 MB

After the conversion into JSON file,`sample_file_osm.json`

JSON output : 194.2 MB

Cleaning the XML file

In the XML file for Melbourne city, we encountered two major problems;

1. Abbreviated street names, for example;

$$< \text{tag k} = \text{"addr:street" v} = \text{"Bay St" / >}$$

2. Abbreviated state names;

$$< \text{tag k} = \text{“addr:state” v=“VIC” / >}$$

In order to clean the XML file, one possible approach is to replace the abbreviated names by their full length counterparts. The auditing has been done in the file; `audit_street.py`. Here, we present few snippets of the Python file to elaborate the cleaning strategy. To begin with, we have created a mapping dictionary, which maps the abbreviated names to their full length counterparts;

```
mapping = { "St": "Street",
  "St.": "Street",
  "Ave":"Avenue",
  "Rd.":"Road",
  "PKWY":"Parkway",
  "Dr.":"Drive",
  "Blvd.":"Boulevard",
  "Ct.":"Court",
  "Ln":"Lane",
  "Centre":"Center"}
```

Next, we construct the following function to execute the mapping. Here, we have shown the update function for street names, but similar approach can be used to update the state names.

```
def update_street_name(name, mapping):
name=name.split()
if name[len(name)-1] in mapping.keys():
                              name[len(name)-1]=mapping[name[len(name)-1]]
                             name = ' '.join(name)

return name
```

In the above function, the input parameter 'name' is a string, such as "Bay St".

Transforming Into JSON Format

Once the input XML file has been cleaned satisfactorily, it is time to convert it into JSON file. To this end, we have used the code from the Preparing for Database Quiz in the MongoDB course, see `clean_audit.py`. From the file `audit_street.py`, we have used the mapping dictionary and the update function. The updating has been done inside the `shape_element` function. A snippet has been presented here;

```
def shape_element(element):
.........
if list_new2[1]=='street':
                     name=update_street_name(tag.attrib['v'], mapping)
                     address['street']=name
...........
return node
```

After the completion of shaping step, the data is loaded into JSON format using `process_map` function.

```
def process_map(file_in, pretty = False):
file_out = "{0}.json".format(file_in)
data = []
with codecs.open(file_out, "w") as fo:
for _, element in ET.iterparse(file_in):
     el = shape_element(element)
     if el:
          data.append(el)
          if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
          else:
              fo.write(json.dumps(el) + "\n")
return data
```

The output is loaded into JSON file, called `sample_file_osm.json`. Here is a snippet.


```
{
"created": {"changeset": "39853241", "timestamp": "2016-06-07T05:33:13Z",
 "uid": "3007", "user": "Hotwheelz", "version": "16"},
"pos": [-37.837462, 144.943356],
"id": "613354748",
"type": "node",
```

```
"name": "Elegant Slax",
"shop": "clothes",
"phone": "61396451271",
"source": "Bay Street",
 "website": "http://www.elegantslax.com.au/",
 "wheelchair": "limited" ,
 "address": {"city": "Port Melboune", "state": "VICTORIA",
 "street": "Bay Street", "country": "Australia", "postcode": "3207",
  "housenumber": "275"}
}
```


Loading into MongoDB

The json file is loaded into MongoDB database, see `load_data.py`.

```
from pymongo import MongoClient
import json
client=MongoClient()
db=client.test  #make a database called test

with open('sample_file_osm.json') as f_in:
#each line is  a JSON array, so we need to loop through
for item in f_in:
     data = json.loads(item)
#sample is  a collection in test database
     db.sample.insert(data)
```

Once the data has been loaded into the database, we can retrieve relevant information using MongoDB query commands.

Querying the Database

Before we begin querying the database, we should make a connection to the running instance of MongoDB.

The querying has been done in the file; `query_mongodb.py`. First, we find number of nodes.

```
node_count=db.sample.count({"type":"node"})

>>> node_count=777517
```

Similary, we can find number of ways;

```
>>> way_count=106850
```

Next, we count number of distinct values for the field `created_by`. This field represents the computer program, which made the changes. Please refer to `http://wiki.openstreetmap.org/wiki/Key:created_by`.

```
distinct_values=db.sample.distinct("created_by")
len(distinct_values)

>>>37
```

Therefore, we have 37 distinct programs which are responsible for the changes in this street map for Melbourne. Using the aggregation pipeline, we can get number of documents added by a particular user. Moreover, we can retrieve the name of the user who has added the largest number of documents.

```
  group_user=db.sample.aggregate([{
    "$group" : {
      "_id" :"$created.user" ,
      "count": { "$sum":1}}},{"$sort":{"count":-1}}])
  list_new=[doc for doc in group_user]
  >>> list_new[0]
  {'_id': 'CloCkWeRX', 'count': 237363} #gives the user with highest number of docs
                                          added
```

In the end, it will be beneficial to find some amenities in the city of Melbourne. First, we would like to know the number of schools, and cafeterias.

```
  docs_school=db.sample.find({"amenity": "school"}).count()
>>> 349

docs_cafe=db.sample.find({"amenity":"cafe"}).count()
>>>301
```

Next, we find the most frequent `amenity` in the city of Melbourne. Similar to user grouping, we can group by the field `amenity`.

```
  group_amenity=db.sample.aggregate([{"$match":{"amenity":{"$exists":1}}},{
    "$group" : {
      "_id" :"$amenity" ,
      "count": { "$sum":1}}},{"$sort":{"count":-1}}])
list_amenity=[doc for doc in group_amenity]
>>> list_amenity[0]
{'_id': 'parking', 'count': 1805}
```

Suggested Extension

It has been observed that some names start with 'St.', which stands for 'Saint'. For example;

$$< \text{tag k} = \text{"name" v} = \text{"St Johns Pre-School"} / >$$

One possible approach can be to replace the abbreviated name by its full name using regular expression.

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

Similar to the concept of replacing the abbreviated names at end of street names, we can use the above regular expression to capture the first word of the text. Next, we can use the mapping dictionary to replace the abbreviated name by its full form.
Alternative approach is to replace this abbreviation using the idea of `re.sub` function.

```
first = name.split(" ")[0]#extract the first word, which might  be an
                                    abbreviation
first = re.sub(r'^\S+\.?\b', 'Saint', first)
#and then reconstruct the string:

name = first + ' '.join(name.split(" ")[1:])
```

Possible Challenge

One possible challenge in implementing the above suggested extension is the ambiguity in the abbreviation. It is not certain that 'St' stands for 'Saint' or not. For example, if we have

$$< \text{tag k} = \text{“name”v} = \text{“St Hotel”} / >$$

we ae not certain that 'St' stands for 'Saint'. If we look closely, we find that it is a pub;

$$< \text{tag k} = \text{“amenity”v} = \text{“pub”} / >$$

But if we look at a different example;

$$< \text{tag k} = \text{“name”v} = \text{“St Mary Magdalen's Parish”} / >$$

If we check the type of amenity,

$$< \text{tag k} = \text{“amenity”v} = \text{“place of worship”} / >$$

Here, we are certain that 'St' refers to 'Saint'. Therefore, we need to be careful before replacing 'St' by their full names, and need to check the nature of amenity. It is also possible that sometimes 'St' in the first can stand for 'Street'. This approach is useful, in the sense that it will provide us more or less complete information about the different types of amenities/services. But this will require more investigation and similar examples can be found after more closer investigation of the XML data set.

Conclusion

In this wrangling exercise, a sample of street map of Melbourne city was examined. The data was cleaned programmatically to convert it into suitable form. After loading the data into MongoDB database, queries have been implemented to retrieve some useful information. Apart from the abbreviated street names and state names, the data is relatively easy to interpret.