Project P2: Data Wrangle OpenStreetMaps Data

Olakunle Kuye

Project Code Snippets (P) = A file that holds project code snippets .

Chosen Map Area

I downloaded the Austin Texas from mapzen metro – extract details found in App 1.

Auditing the data

After downloading the OSM XML data, I parsed it using the code I used for the iterative Parsing exercise found in the Project Code Snippets P 1

I ran the code I used for the Tag Type exercise whereby checks on the dataset were made to determine if they are certain patterns in the tags present in the data.

This is where there are variables declared to search for the following three regular expressions: **lower**, **lower_colon**, and **problemchars**.

lower: This variable searches for strings containing lower case characters

lower_colon: Searches for strings containing lower case characters and a single colon within the string

problemchars: Searches for characters that cannot be used within keys in **MongoDB**.

Code snippets found in P2.

As the image below shows there are a number of tags with multiple children tags that begin with address.

These will be used with the lower_colon regex to find these keys so that it's possible to build a single address document within a larger json document shown in P3.

Here is a partial output and a screen shot from running the code used in the Tag Type exercise:

```
{'lower': 226464, 'lower_colon': 299286, 'other': 10123, 'problemchars': 2}
```

By modifying the code found in P4 used in Tag Type exercise I built a set of unique ids found in the xml, outputting the length of the set that represents the number of users making edits in the chosen map area.

Problems with the Data

The majority of this project will be devoted to auditing and cleaning street names seen within the OSM XML.

Replacing Underscores in the Values

Within the data allot of values for the tags with "barrier", "highway", "amenity" ," source", "shop" and "emergency" have underscores within them and I will amend the data by using the following code :

- A dictionary will be built to hold the values of "k" tags with names such as "barrier", "higher", "amenity", "emergency", "source" and "shop" such as :
  UNDERSCORE_RELATED_TAGS = ["barrier", "highway", "amenity","emergency","source","shop"]

Street Names

Street types used by users in the process of mapping are quite often abbreviated. I will attempt to find these abbreviations and replace them with their full text form. The plan of action is as follows:

The data contained in the OSM XML file will be audited to clean the street types used by users, which are often abbreviated. These abbreviation's will be found and replaced with the full street name.

In order to do this, the following steps will be taken:

- A regex designed to match the last token in a string will be developed, finding the street type in an address.
- I would then proceeded to build a list of expected street types that don't need to be cleaned.
- The XML file will be parsed for tag elements with the **k="addr:street" attributes** .
- A search will be conducted using the regex on the value v attribute of the street name string.
- A dictionary will be built with keys that match the street types in the regex, setting the street names where a specific key was found as a value, this enables the determination of what needs to be cleaned.
- Another dictionary will be built to contain a map from an offending street type. This along with another regex and a function that will be developed and used to match the offending street types will be used to clean the data. The function will return a cleaned string.

Building a regex to match the last token in a string optionally ending with a period and a list of street types to clean found in P5.:

from collections import defaultdict

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

expected_street_types = ["Avenue", "Boulevard", "Commons", "Court", "Drive", "Lane", "Parkway","Place", "Road", "Square", "Street", "Trail"]

The **audit_string** takes in the dictionary of street types, an audit string a regex match and a list of expected street types.

The function will search the string passed to it for a regex and if it finds a match will that is not in the dictionary used to hold expected dictionaries it will be added to it . Code found in P6.:

```
def audit_string(match_set_dict, string_to_audit, regex, expected_matches):
    m = regex.search(string_to_audit)
    if m:
        match_string = m.group()
        if match_string not in expected_matches:
            match_set_dict[match_string].add(string_to_audit)
```

An audit function is defined and it's used to parse and audit street names. The function audits tag elements where **k="addr:street"** and it also matches the tag filter function . The audit function takes in a regex and a list of expected matches. Relaed code found in **P7**.

A portion of the pretty print of the output of the audit function can be seen in P8.

There were a number unexpected abbreviated street types as well as cardinal directions and highway directions. I created a function to replace the abbreviated street types taking a regex to search for, updating a mapping directory. The required code and results summary can be found in **P9**.

```
The keys are replaced where they appear in a string using: bad_streets =
"|".join(map_street_types.keys()).replace('.', '')
```

```
street_type_updater_re = re.compile(r'\b(' + bad_streets + r')\b\.?',
re.IGNORECASE)
```

Results Summary:

```
W US Highway 290 Service Rd => W US Highway 290 Service Road
Saddleback Rd => Saddleback Road
Barley Rd => Barley Road
Open Sky Rd => Open Sky Road
Elderberry Rd => Elderberry Road
N Interstate 35 Frontage Rd => N Interstate 35 Frontage Road
Spicewood Springs Rd => Spicewood Springs Road
```

Cardinal Directions

The cardinal directions North, South, East and West appear to be abbreviated in the data, so using a similar technique to set the correct the abbreviated cardinals by creating a regex to match NSEW at the begging of a string, followed by an optional period.

Using the audit method:

```
cardinal_directions = audit(OSMFILE, is_street_name, cardinal_dir_re)
```

Results Summary:

```
{'E': set(['E 38th 1/2 St.',
           'E 43rd St',
           'E Hwy 290',
           'E Live Oak St',
           'E Main Street',
           'E Oltorf St',
           'E Palm Valley Boulevard',
           'E Riverside Drive']),
 'E.': set(['E. 43rd St.', 'E. North Loop', 'E. University Ave']),
…)}
```

The output shows the cardinals at the beginning street names, but creating an exhaustive mapping will resolve this using:

```
map_cardinal_directions = \
    {
        "E" : "East",
        "E." : "East",
        "N" : "North",
        "N." : "North",
        "S" : "South",
        "S." : "South",
        "W" : "West",
        "W." : "West"
    }
```

Whereby the keys will be replaced anywhere in the string using a regex with the following code :

```
bad_directions = "|".join(map_cardinal_directions.keys()).replace('.', '')
```

```
cardinal_dir_updater_re = re.compile(r'\b(' + bad_directions + r')\b\.?',
re.IGNORECASE)
```

I modified the code used in the **audit.py** for exercise 6, traversing the dictionary applying updates for street types and the appropriate cardinal direction.

Using:

```
    cardinal_directions = audit(OSMFILE, is_street_name, cardinal_dir_re)
    for cardinal_direction, ways in cardinal_directions.iteritems():
        if cardinal_direction in map_cardinal_directions:
            for name in ways:
                better_name = update(name, map_street_types,
street_type_updater_re)
                best_name = update(better_name, map_cardinal_directions,
cardinal_dir_updater_re)
                print name, "=>", better_name, "=>", best_name
```

Results Summary:

```
E. University Ave => E. University Avenue => East University Avenue
E. 43rd St. => E. 43rd Street => East 43rd Street
E. North Loop => E. North Loop => East North Loop
E Oltorf St => E Oltorf Street => East Oltorf Street
E Live Oak St => E Live Oak Street => East Live Oak Street
```

Preparing and Loading Data into MongoDB

To load data into MongoDB I transformed the data.

The rules I chose included the following:

To only process two types of top level tags “node” and “way”.
Attributes except from created: “version, changeset, timestamp, user, uid” should be added a key.
The longitudes and latitudes will be added to an array “pos”, to be used in a geospatial indexing, ensuring the values inside the “pos” are floats and not strings.
Problematics characters in the second level of the tag “k” would be ignored.
The dictionary named **TIME_AND_USER_TAGS** in the second tag level with values that starts with “addr” but containing “:” should be added.
The presence of a second tag containing character “:” should be ignored .the character separates the type and direction of a street.

For the transformation will make use of a function named **modify_element** that processes an element. Within the function there is an update feature to using the regex and mapping dictionaries to clean addresses.

The function will also make use of the **UNDERSCORE_RELATED_TAGS** dictionary to determine what k tag values to set by removing the underscore present using code found in **P10**.

Parse XML and write to JSON
To parse the xml and write to json I used the found in **P11**.

Overview of the Data

Data Size

To determine the size of the downloaded file using:

```
import os
print "The downloaded file is {}
MB".format(os.path.getsize(‘D:\\austin_texas.osm’)/1.0e6) # convert from bytes to
megabytes
```

***Results* : 176.584186 MB**

To determine the size of the transformed json file using:

```
print "The json file is {} MB".format(os.path.getsize(‘D:\\austin_texas.osm’+
".json")/1.0e6) # convert from bytes to megabytes
```

***Results* : 256.771823 MB**

The diagram in App 2 in the appendix shows the json file being imported into MongoDB.

The diagram in App 3 in the appendix shows the json file being imported into MongoDB.

Number of Documents

Using the following bit of code to connect to the MongoDB database, I was able to determine the number of documents.

```
db_name = "osm"

client = MongoClient('192.168.255.133:27017')
db = client[db_name]

collection = db.austin_texas

print collection.count()
```

Result: 848323

Number of Unique Users

The number of unique users can be determined by modifying the previous code to include:

```
Print len(collection.distinct('created.user'))
```

Result: 910

Top Contributing User

The top contributing user was known by running the following code :

```
db.austin_texas.aggregate([{"$group" : {"_id" : "$created.user", "count" : {"$sum"
: 1}}}, {"$sort" : {"count" : -1}},{"$limit" : 1}])

{ "_id" : "woodpeck_fixbot", "count" : 241116 }
```

Number of Nodes and Ways

The number of nodes and ways can be known by running the following code in the database (within MongoDB) :

```
aggregate({"$group" : {"_id" : "$type", "count" : {"$sum" : 1}}})['result']
```

Nodes Results :

```
{ "_id" : "node", "count" : 768631 }
```

Way Results :

```
{ "_id" : "way", "count" : 79692 }
```

Number of Documents Containing a Street Address

To determine the number of containing a street address, I modified the previous code (within MongoDB) to use:

```
db.austin_texas.find({"address.street" : {"$exists" : 1}}).count()
```

Results: 2014

Cleaned Zip Codes

To display a list of cleaned zip codes I used the following code (within MongoDB) :

```
db.austin_texas.aggregate([{"$match" : {"address.postcode" : {"$exists" : 
1}}},{"$group" : {"_id" : "$address.postcode", "count" : {"$sum" : 1}}}, {"$sort" 
: {"count" : -1}}])
```

Results Summary :

```
{ "_id" : "78704", "count" : 63 }
{ "_id" : "78705", "count" : 54 }
{ "_id" : "78757", "count" : 48 }
```

Top 5 Most Common Cities

To determine the top five cites most common using the following code (within MongoDB) :

```
db.austin_texas.aggregate([{"$match" : {"address.city" : {"$exists" : 1}}}, 
{"$group" : {"_id" : "$address.city", "count" : {"$sum" : 1}}}, {"$sort" : 
{"count" : -1}}, {"$limit" : 5}])
```

Results Summary :

```
{ "_id" : "Austin", "count" : 541 }
{ "_id" : "Round Rock", "count" : 65 }
{ "_id" : "Austin, TX", "count" : 48 }
```

Top 10 Amenities

To determine the top ten amenities using : db.austin_texas.aggregate([{"$match" : {"amenity" : {"$exists" : 1}}}, {"$group" : {"_id" : "$amenity", "count" : {"$sum" : 1}}}, {"$sort" : {"count" : -1}}, {"$limit" : 10}])

Results Summary :

```
{ "_id" : "parking", "count" : 1845 }
{ "_id" : "restaurant", "count" : 684 }
{ "_id" : "waste_basket", "count" : 591 }
```

Top Religions with Denominations

Top religions with denominations can be determined by running the following code in mongodb to give:

```
db.austin_texas.aggregate([{"$match" : {"amenity" : "place_of_worship"}},{"$group"
: {"_id" : {"religion" : "$religion", "denomination" : "$denomination"}, "count" :
{"$sum" : 1}}},{"$sort" : {"count" : -1}}])
```

Results Summary:

```
{ "_id" : { "religion" : "christian" }, "count" : 196 }
{ "_id" : { "religion" : "christian", "denomination" : "baptist" }, "count" : 108
}
{ "_id" : { "religion" : "christian", "denomination" : "lutheran" }, "count" : 30
}
```

Top 10 Leisures

The top 10 leisure within the data can be determined using

```
db.austin_texas.aggregate([{"$match" : {"leisure" : {"$exists" : 1}}}, {"$group" :
{"_id" : "$leisure", "count" : {"$sum" : 1}}}, {"$sort" : {"count" : -1}},
{"$limit" : 10}])
```

Results Summary :

```
{ "_id" : "pitch", "count" : 828 }
{ "_id" : "park", "count" : 421 }
{ "_id" : "playground", "count" : 104 }
{ "_id" : "sports_centre", "count" : 70 }
```

About the Dataset:

I believe the data is well formed and structured making it suitable for extension whereby user will be able to add reviews with regards to bicycle routes, restaurants and schools etc.
The map could exist as an application programming interface that could be embed into another application consisting of a function or layer that could show the most recently updated parts.

Appendix

App 1

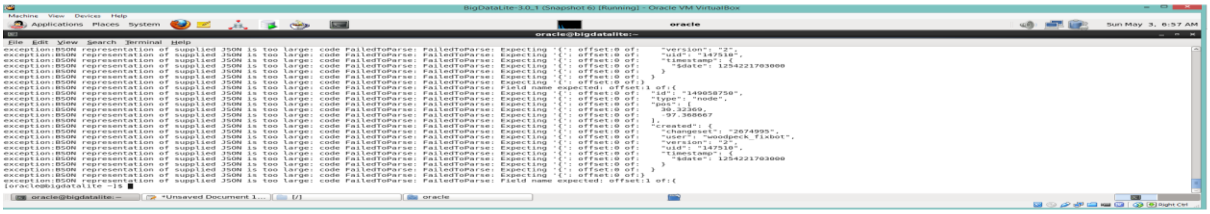
http://www.openstreetmap.org/export#map=13/30.3136/-97.7027&layers=CND

Longtitude : 30.3846, 30.2426

Latitude : -97.8557, -97.5497

The map above shows the cycle path alongside the Lake Austin.

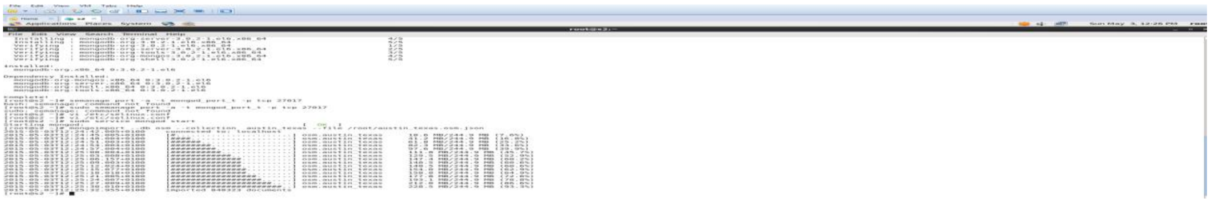
App 2



BigDataLite-3.0_1 (Snapshot 6) [Running] - Oracle VM VirtualBox
Machine View Devices Help
Applications Places System
oracle
Sun May 3, 6:57 AM
oracle@bigdatalite:~
File Edit View Search Terminal Help
oracle@bigdatalite:~
*Unsaved Document 1...
[/]
oracle
Right Ctrl

```
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "version": "2",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "uid": "147510",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "timestamp": {
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:       "$date": 1254221703000
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     }
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:   }
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:}
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Field name expected: offset:1 of:{
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:   "id": "149058750",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:   "type": "node",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:   "pos": [
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     30.32369,
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     -97.368667
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:   ],
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:   "created": {
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "changeset": "2674995",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "user": "woodpeck_fixbot",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "version": "2",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "uid": "147510",
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     "timestamp": {
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:       "$date": 1254221703000
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:     }
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:   }
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Expecting '{': offset:0 of:}
exception:BSON representation of supplied JSON is too large: code FailedToParse: FailedToParse: Field name expected: offset:1 of:{
[oracle@bigdatalite ~]$ 
```

App 3



File Edit View VM Tabs Help
Home
s2
Applications Places System
Sun May 3, 12:26 PM
root@s2:~
File Edit View Search Terminal Help

```
  Installing  : mongodb-org-server-3.0.2-1.el6.x86_64                    4/5
  Installing  : mongodb-org-3.0.2-1.el6.x86_64                           5/5
  Verifying   : mongodb-org-3.0.2-1.el6.x86_64                           1/5
  Verifying   : mongodb-org-server-3.0.2-1.el6.x86_64                    2/5
  Verifying   : mongodb-org-tools-3.0.2-1.el6.x86_64                     3/5
  Verifying   : mongodb-org-mongos-3.0.2-1.el6.x86_64                    4/5
  Verifying   : mongodb-org-shell-3.0.2-1.el6.x86_64                     5/5

Installed:
  mongodb-org.x86_64 0:3.0.2-1.el6

Dependency Installed:
  mongodb-org-mongos.x86_64 0:3.0.2-1.el6
  mongodb-org-server.x86_64 0:3.0.2-1.el6
  mongodb-org-shell.x86_64 0:3.0.2-1.el6
  mongodb-org-tools.x86_64 0:3.0.2-1.el6

Complete!
[root@s2 ~]# semanage port -a -t mongod_port_t -p tcp 27017
bash: semanage: command not found
[root@s2 ~]# sudo semanage port -a -t mongod_port_t -p tcp 27017
sudo: semanage: command not found
[root@s2 ~]# vi /etc/selinux.conf
[root@s2 ~]# vi /etc/selinux.conf
[root@s2 ~]# sudo service mongod start
Starting mongod:                                           [  OK  ]
[root@s2 ~]# mongoimport --db osm --collection austin_texas --file /root/austin_texas.osm.json
2015-05-03T12:24:42.005+0100    connected to: localhost
2015-05-03T12:24:45.005+0100    [#.......................] osm.austin_texas    18.6 MB/244.9 MB (7.6%)
2015-05-03T12:24:48.003+0100    [####....................] osm.austin_texas    41.2 MB/244.9 MB (16.8%)
2015-05-03T12:24:51.003+0100    [######..................] osm.austin_texas    61.8 MB/244.9 MB (25.2%)
2015-05-03T12:24:54.004+0100    [########................] osm.austin_texas    82.3 MB/244.9 MB (33.6%)
2015-05-03T12:24:57.004+0100    [#########...............] osm.austin_texas    97.6 MB/244.9 MB (39.9%)
2015-05-03T12:25:00.004+0100    [##########..............] osm.austin_texas    111.8 MB/244.9 MB (45.7%)
2015-05-03T12:25:03.008+0100    [############............] osm.austin_texas    129.5 MB/244.9 MB (52.9%)
2015-05-03T12:25:06.157+0100    [##############..........] osm.austin_texas    147.4 MB/244.9 MB (60.2%)
2015-05-03T12:25:09.003+0100    [##############..........] osm.austin_texas    148.5 MB/244.9 MB (60.6%)
2015-05-03T12:25:12.024+0100    [##############..........] osm.austin_texas    148.5 MB/244.9 MB (60.6%)
2015-05-03T12:25:15.077+0100    [###############.........] osm.austin_texas    154.6 MB/244.9 MB (63.1%)
2015-05-03T12:25:18.010+0100    [###############.........] osm.austin_texas    158.8 MB/244.9 MB (64.9%)
2015-05-03T12:25:21.005+0100    [#################.......] osm.austin_texas    177.8 MB/244.9 MB (72.6%)
2015-05-03T12:25:24.007+0100    [##################......] osm.austin_texas    193.1 MB/244.9 MB (78.8%)
2015-05-03T12:25:27.005+0100    [####################....] osm.austin_texas    212.8 MB/244.9 MB (86.9%)
2015-05-03T12:25:30.010+0100    [######################..] osm.austin_texas    228.5 MB/244.9 MB (93.3%)
2015-05-03T12:25:32.955+0100    imported 848323 documents
[root@s2 ~]# 
```