Heather Capell, Data Analyst Nanodegree (July 2015 Cohort)
P3 Project Re-Submission (1/10/16): OpenStreetMap Project

Map Area: Austin, Texas metropolitan area (used a 1/20th subset to keep to 50MB size)

1. Problems Encountered

a. Need for expanded Mapping pairs for Update_Name

During the audit and cleansing process, I had problems initially with a not robust enough list of Mapping pairs to use with the Update_Name function. Iterating on that function many times, I was able to discover additional street abbreviations being used in the OSM data for Austin, and could use these to expand upon the list of Street Names that would need to be improved/cleaned.

```
mapping = { "St.": "Street",
            "St": "Street",
            "Ave.": "Avenue",
            "Ave": "Avenue",
            "Rd.": "Road",
            "Rd": "Road",
            "Hwy": "Highway",
             …etc. }
```

b. Update_Name was evaluating parts of words rather than whole word

I had trouble getting the Update_Name function to only focus on the last word in the Street Name string, which resulted in parts of words be used in the Mapping. For example, the word 'Drive' would be replaced in the Update_Name function via the Mapping for Dr > Drive, to result in 'Driveive'. To resolve this, I added a split string expression and chose the last word in the string to match against the Mapping list.

Early versions of the cleansing code yielded situations like this:

{"created": {"uid": "3370181", "changeset": "35510650", "version": "1", "user": "ccjjmartin_atxbuildings", "timestamp": "2015-11-22T16:34:47Z"}, "pos": null, "height": "12.9", "visible": null, "address": {"street": "Zambia Driveive", "housenumber": "2732"}, "type": "way", "id": "381982558"}

So I updated the code to split the string into separate words and then match only on the last word, so that match would not occur on parts of words. I used the last word because that is the typical location on an American address for the generic street types that are easiest to identify and cleanse. In some cases, the string for "street" ends with a directional letter or word (N or North, for example), and for that scenario the code

would not review/match on the street type, however (more on that below). Updated code:

```
def update_name(name, mapping):
    for v in mapping:
        if re.match(v,name.rsplit(None,1)[-1]):
            name=name.replace(v,mapping[v])
    return name
```

c. Evaluating Zip Codes

I made additional audits on zip code (postcode), and found that the vast majority followed the acceptable 5-digit or 9-digit numeric format and were in the '787xx' and 786xx' range appropriate for the Austin area. However, some of the documents had a 'TX ' (or 'Texas') as a prefix, which needed to be cleansed.

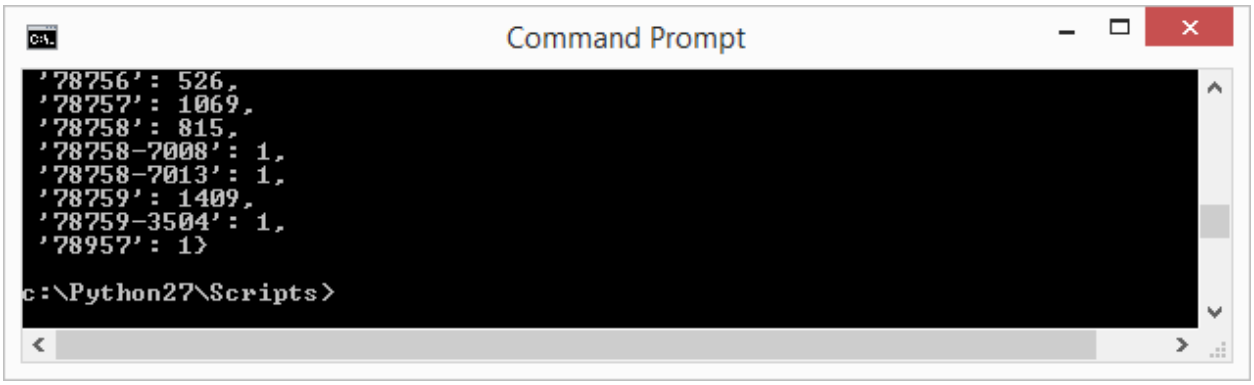
Here is the code used to audit the zip code values:

```
def get_postcode(element):
    if (element.attrib['k'] == "addr:postcode"):
        postcode = element.attrib['v']
        return postcode
```



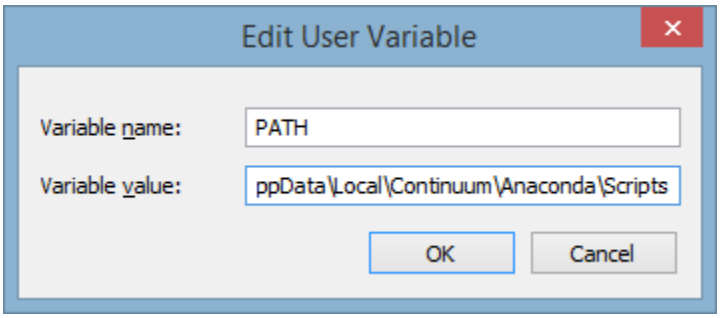
And to cleanse the zip code values, I used the following replacement:

```
def update_postal(postcode):
    for v in badzip:
        if postcode is None:
            continue
        else:
            if re.match(v,postcode):
                postcode = postcode.replace(v,badzip[v])
    return postcode
```



d. Installing Python and MongoDB

Getting both Python and MongoDb installed on my computer involved a lot of troubleshooting. I learned that I needed to have the file path/location of the executable files for either (including for items like the PIP used to install packages) saved as a PATH Environmental Variable, in order to have the commands available at a Command Prompt. Some other programs I had loaded (not related to this program), such as SQL and Oracle, had PATH Variables set that were interfering with this working properly, as was an Anaconda version of Python that was located in a different path/location that another simpler version of Python. I removed some of the PATH Variables to get this functionality working properly and finally have easier access to running Python and Mongo commands.



e. Sample Project MongoDB commands in error

Once MongoDB was installed and the database imported, I had trouble running some of the basic queries shown in the sample project. In particular, all the queries that would show the top number of something, such as users with most contributions or amenities with the highest counts. A piece of the query had to be modified for all that included a sort command in order to get the counts in descending order, using {“count”: -1} instead of {“count”: 1}.

```
> db.aus.aggregate([{"$group":{"_id":"$created.user",
"count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":5}])
```

2. Data Overview

File Sizes

```
austin_texas.osm ..............  1,065.6 MB [original file from OpenStreetMap)
austin_texas_smaller.osm ..........53.9 MB [subset of file, 1/20th of orig.]
austin_texas_smaller.osm.json .... 70.2 MB
```

Number of Documents

```
> db.aus.count()
272,248
```

Number of Nodes

```
> db.aus.find({"type":"node"}).count()
246,666
```

Number of Ways

```
> db.aus.find({"type":"way"}).count()
25,582
```

Number of Unique Users

```
> db.aus.distinct("created.user").length
547
```

Top 5 Contributing Users

```
> db.aus.aggregate([{"$group":{"_id":"$created.user",
"count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":5}])

{ "_id" : "patisilva_atxbuildings", "count" : 85,011}
{ "_id" : "ccjjmartin_atxbuildings", "count" : 64,801}
{ "_id" : "ccjjmartin__atxbuildings", "count" : 46,125}
{ "_id" : "jseppi_atxbuildings", "count" : 13,010}
{ "_id" : "woodpeck_fixbot", "count" : 11,506}
```

Number of Users with Only 1 Contribution

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

{"_id":1,"num_users": 159}
```

3. Additional Data Exploration and Ideas for Improvement

Top 6 Amenities

```
> db.aus.aggregate([{"$match":{"amenity":{"$exists":1}}},
{"$group":{"_id":"$amenity",
"count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":6}])

{"_id" : "parking","count": 98}
{"_id" : "restaurant", "count" : 40}
{"_id" : "place_of_worship", "count" : 29}
{"_id" : "waste_basket", "count" : 27}
{"_id" : "school", "count" : 26}
{"_id" : "fast_food", "count" : 25}
```

Top 5 Cuisines

```
> db.aus.aggregate([{"$match":{"amenity":{"$exists":1},
"amenity":"restaurant"}}, {"$group":{"_id":"$cuisine",
"count":{"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":5}])

{ "_id" : null "count" : 23}
{ "_id" : "mexican", "count" : 5}
{ "_id" : "indian", "count" : 1}
{ "_id" : "italian", "count" : 1}
{ "_id" : "asian", "count" : 1}
```

Observations

An observation about the amenities listed is that the top amenities relate to parking and food establishments -- not at all surprising in a college town. Of the food establishments, Mexican is the top one (other than Null restaurant types), which lines up with my own personal experience of Austin.

Another observation is that 4 of the top 5 contributors were associated with an "ATX_buildings" OSM login. After doing some Google searches, I learned that a project is now underway to merge the City of Austin public records for 2013 Building Footprints and monthly Address Points datasets into the OpenStreetMap for Austin, using community volunteers familiar with the Austin area. This project is similar to one completed in New York City in June 2014, where data for 1 million buildings and 900,000 addresses were imported to OpenStreetMap. Austin's dataset impressively comprises around half that size: 585,000 buildings. There were some interesting lessons learned from the New York City project that were referenced in the Austin project wiki site. Some of the key issues for NYC involved duplicate Nodes (need to be merged), duplicate addresses for the same building (need to be validated by other data sources), and how best to handle Skyways (to show a Way that is above ground).

Ideas to Improve Upon

One item that could be improved in my code is how to deal with street types that are not found in the last word of the "street" string. Many of the "directional" parts of street addresses, such as North, South, etc. appear at the end of the street string. They also frequently appear at the beginning of the street string, and code could be added to check/match on these directional abbreviations to replace with the full word. However, any cleansing to address this would need to avoid updating or cleansing N, S, E, W abbreviations if they are found within a word—i.e., we would not want to replace 'E' in 'Eddison Street' with 'East', because that would yield 'Eastddison Street.'

Relating to the above issue, a nice improvement to OpenStreetMaps would be to have a new tag type for Street Types, since this data is currently contained in a tag that has the Street Name as well—a key tenet of master data management being that storing two kinds of data in a single field is not a best practice for usability (portability) and data quality. That improvement to OSM could have drawbacks, though, because many data sources that import into OSM may not have the information split out by Street Name and Street Type into different fields/tags, so could be cumbersome to provide information to OSM in this format.

4. Conclusions

I was surprised that that there was not more comprehensive data currently in the Austin, Texas OpenStreetMap, but also very pleased to learn about the community project that launched to bring more date to OSM. Since Austin has such a culture of activism and is also a university town, I was expecting more contributions from sources independent from the ATX_buildings project. The total number of contributors was 547, with 129 making only one contribution, which seems on the low side for a city of half a million people. Nonetheless, I am excited about the prospect of bringing the City of Austin public data into a platform that can be expected to have much more widespread use. For the similar New York City OSM data import project, I learned that the tech startup FourSquare used this public domain data as part of its successful business model. It is my hope that Austin can have similar successes.

As a final takeaway, there was an article in OSM about guidelines for OSM import projects, which very closely mirrored my own experience with this course project. The top of the list was a note that importing takes a lot of time, so to make sure that you have the time to commit, and the second point was that you need to be ready to continuously iterate on your conversion/upload scripts as well as on data already loaded. My data wrangling project took many, many iterations on the code for cleansing alone, and more than one try at the upload process to identify issues and improve the whole process. Although I have heard frequently that data wrangling is at least half of a data analyst or data scientist's job, this course project really brought that aspect home because I was living a day (or a month!) in the life of a Data Analyst.