Data Analyst Nanodegree

Usage

I've organized my submission into the following structure:

- cleaning - holds the Python code used to audit, parse, and wrangle the data
  - queries - holds the queries.json file, which has a list of all the queries I use to audit the data once it has been inserted into MongoDB
- data - holds the various .osm and .json data files
- documents - holds project description

I have created a driver program called wrangler.py - it is located in */cleaning* and if you want to follow my steps, you can just run the driver and go through the prompts. Everything I do below is actually done using the driver (save for some of the more basic MongoDB queries and importing the JSON data into MongoDB).

Auditing Street Names

I first try to get a feel for the sorts of street name corrections I'll need to do. In my driver, I call audit.py to examine the various street names. Compared to the list of expected tokens in Lesson 6, there were many more conversions that needed to be made in my file: for example, "expway" and "expwy" both had to be converted into "Expressway".

For the majority of cases, the Lesson 6 mapping plus a few additions was enough. For example:

```
Updating  E. 43rd St.
    Now  East 43rd Street
```

The cleaner correctly fixed the abbreviations. However, a few problems remained.

```
Updating  N IH-35
    Now  North IH-35
```

In this case, the IH is not fixed. I had never actually seen the abbreviation "IH" before, but it turns out that it stands for "Interstate". There were a lot of entries with "Interstate 35" in them, and a quick Google search of "IH-35" confirmed my suspicions that this was just another way to abbreviate "Interstate" (I'm assuming that it stands for "Interstate Highway").

Also, there is also the case of "FM", or "Farm to Market" roads (along with "RM", or "Ranch to Market" roads). While not unique to Texas, they are primarily found in this state and are an officially designated type of roadway. They allow farmers and ranchers in agricultural/rural areas to push their goods to centralized marketplaces. For example:

```
Updating  FM 685
    Now  FM 685
```

In the case of "FM" and "RM" roads, fixing them was a simple matter of adding more mappings. Fixing "IH" required a little more work, simply because it came in a few forms. For example:

```
Updating  N I H 35
    Now  North Interstate H 35
```

```
Updating  North IH 35
    Now  North IH 35
```

were also found. These two cases, though, are simple enough - I just added two more mappings, one for “IH” and one for “H”. I used Python’s replace() function to fix cases of “IH-“.

I also fixed something that was bugging me: suite numbers. Post-fix, it looks something like this:

```
Updating  N. Lamar suite#L131
    Now  North Lamar Suite L131
```

Much better than before.

There were still more problems, however.
For example,

```
FM1431 => FM1431
I35 => I35
S Bell Blvd., Suite 301 => South Bell Blvd., Suite 301
Shoal Creek Blvd, Bld 3 => Shoal Creek Blvd, Bld 3
12200 Bee Cave Pkwy, Bee Cave, TX 78738 => 12200 Bee Cave Pkwy, Bee Cave, Texas 78738
```

For the first case, where the street-type abbreviation is not separated from its number, I used Python string methods to correct the issue.
For the second case, the problem was that the commas messed up my parsing, as I only stripped periods at this point. I decided to use the translate() function to take care of punctuation.
And I added another mapping to correct the last issue: “Bld” to “Building”.`

Post Fix

```
FM1431 => Farm to Market Road 1431
I35 => Interstate 35
S Bell Blvd., Suite 301 => South Bell Boulevard Suite 301
Shoal Creek Blvd, Bld 3 => Shoal Creek Boulevard Building 3
12200 Bee Cave Pkwy, Bee Cave, TX 78738 => 12200 Bee Cave Parkway Bee Cave, Texas
78738
```

Regarding my decision to remove all commas outright - in the end, they are not the most vital thing in the world, though I can think of a few cases where it might pose a problem in comprehension.

Notice how the last line is a little confusing with the repeated Bee Cave (which is the name of the town as well as of the parkway). It could be even more confusing if, for example, two different addresses had “Bee Cave Parkway West, Bee Cave” and “Bee Cave Parkway, West Bee Cave”. My code would have converted these into the same thing: “Bee Cave Parkway West Bee Cave”, and some information regarding the exact street address would likely be lost.

I think it is still a worthwhile sacrifice (well, not a sacrifice in my eyes but a gain, because I dislike having commas in data) - and in any case, if others tried to use this cleaned data in some application or project, they would not be using the street name by itself to determine an exact street address anyways.

Characteristics

```
171 MB - austin_texas.osm
179 MB - austin_texas.json
```

I first ran mapparser.py's count_tags() on my austin_texas.osm file. This gives me the following:

```
{'bounds': 1,
 'member': 13013,
 'nd': 904069,
 'node': 781465,
 'osm': 1,
 'relation': 1288,
 'tag': 537658,
 'way': 81093}
```

Obviously, it's a big set of data. Of interest are the 80,000 ways and nearly 800,000 nodes. This is before inserting them into MongoDB, however.

Then, I parsed the tags using tags.py, giving me the following:

```
{'lower': 227288, 'lower_colon': 300162, 'other': 10206, 'problemchars': 2}
```

So we know from the running count_tags() that there are 537,658 total tags, and the numbers here add up correctly. 42.3% are just plain lowercase, 55.8% are lowercase with colons, 1.9% are unclassified, and only 2 of the total, which is 0.00037%, are problematic tags for mongoDB.

I found the two tags under problemchars - the culprit was the use of spaces. The two problem keys are shown below:

```
Hardware Store
moving boxes
```

Next up, I try to find the number of users users who have been contributing to the OpenStreetMap of Austin, Texas.

```
804
```

So, there are several hundred users contributing to the map. As we will see later, though, the distribution of contributions is enormously skewed due to the presence of some bots, which I will identify later.

After inserting the documents in MongoDB, we get the following characteristics.

Number of Documents

```
> db.austin_texas.find().count()
862558
```

Number of "nodes"

```
> db.austin_texas.find({"type":"node"}).count()
781461
```

Number of "ways"

```
> db.austin_texas.find({"type":"way"}).count()
```

```
81086
```

Queries

After reshaping all the data into JSON format and importing them into MongoDB, I ran some queries. I edited the function Udacity provided to name the output file - I didn't like the ".osm.json". Thus, the output file is "austin_texas.json".

I created a file called queries.json that contains JSON docs in the following format:

```
    "get_created_by": {"pipeline" : [
        {"$match": {"created_by": {"$exists": true}}},
        {"$group": {"_id": {"created_by": "$created_by"},
                    "count": {"$sum": 1}}},
        {"$sort": {"count": -1}},
        {"$limit": 10}
        ]
    }
```

I do this to make it easier for me to run my queries from inside my driver. Selecting the right options programmatically lists all the queries I have by name (in this case, "get_created_by" is the name of the query), and then can select the one you wish to run on the collection.

get_created_by

This query shows us the highest contributors to Austin's OSM data. As you can see, Merkaartor and Potlatch dominate the total contributions - these are bots (Merkaartor has two version, 0.12 and 0.13, that have contributed in a major fashion in this data set).

```
        {"$match": {"created_by": {"$exists": true}}},
        {"$group": {"_id": {"created_by": "$created_by"},
                    "count": {"$sum": 1}}},
        {"$sort": {"count": -1}},
        {"$limit": 10}
```

Results are:

```
{u'ok': 1.0,
 u'result': [{u'_id': {u'created_by': u'Merkaartor 0.12'}, u'count': 4702},
            {u'_id': {u'created_by': u'Potlatch 0.10f'}, u'count': 2515},
            {u'_id': {u'created_by': u'Merkaartor 0.13'}, u'count': 995},
            {u'_id': {u'created_by': u'xybot'}, u'count': 35},
            {u'_id': {u'created_by': u'Potlatch 0.10b'}, u'count': 27},
            {u'_id': {u'created_by': u'polyshp2osm-multipoly'}, u'count': 8},
            {u'_id': {u'created_by': u'OSM Fixer'}, u'count': 7},
            {u'_id': {u'created_by': u'Potlatch 0.9b'}, u'count': 6},
            {u'_id': {u'created_by': u'Potlatch 0.9a'}, u'count': 5},
            {u'_id': {u'created_by': u'Potlatch 0.9c'}, u'count': 2}]}
```

get_unique_users

This queries tells us the number of unique users that exist in the data.

```
        {"$group" : {"_id": {"uid" : "$created.uid"}}},
        {"$group" : {"_id": "Number of unique users",
```

```
                        "count": {"$sum": 1}}}
```

Results are:

```
{u'ok': 1.0, u'result': [{u'_id': u'Number of unique users', u'count': 900}]}
```

get_unique_amenities

This queries tells us the number of unique amenities that exist in the data.

```
        {"$group" : {"_id": {"amenity": "$amenity"},
                     "count": {"$sum": 1}}},
        {"$group": {"_id": "Number of unique amenities",
                    "count": {"$sum": 1}}}
```

Results are:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'Number of unique amenities', u'count': 84}]}
```

get_num_streets

This queries tells us the number of documents that have a street name in them.

```
        {"$match" : {"address.street": {"$exists": true}}},
        {"$group": {"_id": "Number of docs with streets",
                    "count": {"$sum": 1}}}
```

Results are:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'Number of docs with streets', u'count': 2010}]}
```

get_most_popular_amenities

This queries tells us the number of documents that have a street name in them.

```
        {"$match" : {"amenity": {"$exists": true}}},
        {"$group" : {"_id": {"amenity": "$amenity"},
                     "count": {"$sum": 1}}},
        {"$sort" : {"count": -1}},
        {"$limit" : 5}
```

Results are:

```
{u'ok': 1.0,
 u'result': [{u'_id': {u'amenity': u'parking'}, u'count': 1857},
            {u'_id': {u'amenity': u'restaurant'}, u'count': 690},
            {u'_id': {u'amenity': u'waste_basket'}, u'count': 591},
            {u'_id': {u'amenity': u'school'}, u'count': 574},
            {u'_id': {u'amenity': u'fast_food'}, u'count': 510}]}
```

get_fast_food

This queries tells us the number of documents that have a street name in them.

```
        {"$match" : {"amenity": "$exists",
                     "amenity": "fast_food"}},
        {"$group" : {"_id": "$name",
                     "count": {"$sum": 1}}},
        {"$sort" : {"count": -1}},
        {"$limit" : 5}
```

Results are:

```
{u'ok': 1.0,
```

```
 u'result': [{u'_id': u'Whataburger', u'count': 31},
             {u'_id': u"McDonald's", u'count': 30},
             {u'_id': u'Subway', u'count': 25},
             {u'_id': u'Taco Bell', u'count': 23},
             {u'_id': u'Sonic', u'count': 19}]}
```

get_fuel

This queries tells us the most popular gas stations.

```
        {"$match" : {"amenity": "$exists",
                     "name": {"$exists": true},
                     "amenity": "fuel"}},
        {"$group" : {"_id": "$name",
                     "count": {"$sum": 1}}},
        {"$sort" : {"count": -1}},
        {"$limit" : 5}
```

Results are:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'Shell', u'count': 79},
             {u'_id': u'Exxon', u'count': 44},
             {u'_id': u'Chevron', u'count': 30},
             {u'_id': u'Texaco', u'count': 29},
             {u'_id': u'Valero', u'count': 26}]}
```

Improvements and Uses

I still wish I could have go back and implement a standardized format for all the street names, such as “<number> <cardinal direction> <street name> <street type>”. This would make things much more orderly - what I could do is implement a Python data structure that has all of these fields and parse through each street name to see if the tokens fit into any of these categories. It would make the data processing more complicated, however, and might not be robust for more varied datasets. Problems could arise when encountering ambiguities, such as street names that contain “Texas” (such as “South Texas Road) and highways that are called “Texas 27 South” (which would be a state highway).

Conclusion

All in all, Texas is an interesting place to analyze. It’s pretty cool to see some of the subtle regional differences within the United States - up in the Northeast we don’t have “Whataburger”, while in Austin it seems to be the most popular fast food dig, and we rarely see a “Texaco” gas station, while in Texas there seem to be quite a few. There are, of course, more ways in which the data could be cleaned, and many more ways that the data could be used. OpenStreetMap data could be used to compare differences in the extent of national store chains - for example, Walmart could use it to see whether there are any areas that lack suitable coverage by a local Walmart. I hope to explore these use cases in further projects.