Wrangling Data in SQL: OpenStreetMap Edmonton and Area

Map Area:

I chose to complete this project on the Edmonton, Alberta, Canada area, accessible at:

- OpenStreetMap
- MapZen

Challenges Encountered during Wrangling:

I started investigating the data on a small subset of the original file, using sample.py and a k of 100 to create and osm file that was 1% of the original dataset (which is 825.5 MB). The resulting sample_k100.osm file is 8.3 MB.

Geobase Tags:

Running tags.py against the sample osm revealed that there were no tags with problem characters, but there were some that were classified as "other". On further investigation, these tags had capitalization in them, and were all coming from Geobase tags: "geobase:acquisitionTechnique" and "geobase:datasetName". The source of these tags was the Canadian Geospatial Data Sets. Ultimately, since the data contained in these tags was irrelevant to my ultimate goals of investigation, I decided to ignore them when parsing the osm file to the csv files.

Tag Names:

After this, I decided to take a look at what tag names were present in my dataset. There are 65 total possible "k" values, including: 'addr:city', 'addr:housenumber', 'addr:interpolation', 'addr:postcode', 'addr:province', 'addr:street', 'amenity', 'attribution', 'barrier', 'bicycle', 'bridge', 'building', 'building:levels', 'cables', 'contact:phone', 'created_by', 'crossing', 'electrified', 'fixme', 'foot', 'footway', 'frequency','gauge', 'geobase:acquisitionTechnique', 'geobase:datasetName', 'geobase:uuid', 'highway', 'intermittent', 'is_in', 'lamp_type', 'landuse', 'lanes', 'lanes:backward', 'lanes:forward', 'layer', 'leisure', 'lit', 'maxspeed', 'name', 'natural', 'office', 'oneway', 'opening_hours', 'phone', 'place', 'population', 'power', 'railway', 'ref', 'resource', 'restriction', 'service', 'shop', 'source', 'sport', 'surface', 'tunnel', 'type', 'voltage', 'water', 'waterway', 'website', 'wikidata', and 'wikipedia'.
To decrease the size of the resulting csv files and time needed to run data.py on the osm file, I decided to "ignore" tags that I deemed irrelevant to my investigation by including an ignore_list variable in my data.py file, comparing tags to that list, and ignoring the tag if it was part of that list. I decided to add 19 tags to that list.

Street Names:

As with the example provided in the OpenStreetMap Case Study in the Udacity classroom, there were many cases of inconsistency in the naming of streets, e.g. “Blvd” vs “Boulevard” and “ST”, “St.” and “street” rather than “Street”. To clean this data, I used the starter code from the case study, and added in some additional words that I would expect in the street name and updated the mapping variable to reflect the variations that I encountered.

Another problem that I encountered was that street names were tagged under both “addr:street” and “name” tags. To clean this up, I converted any “name” tag that was actually a street name using this function, which compares the content of the name tag to the expected words in the street name:

```
# some street names are listed under "name" instead of "addr:street"
if tag.attrib['k'] == "name":
    for word in tag.attrib['v'].split(' '):
        if word in expected:
            tag.attrib['k'] = "addr:street"
            break
```

Bus Stops:

When looking through the “name” tag, I decided to investigate what some of the 4-digit values referred to. What I came across was this:
<tag k="name" v="2210" />
<tag k="highway" v="bus_stop" />
So bus stops are being listed as highways… and the bus stop number is being listed as a name. I wondered if maybe these bus stops were located on a highway, and investigated a few using the Edmonton Transit Website, but this was not the case, many were residential or transit center stops. So, I decided to change the tag to “amenity” within my audit function if the value was “bus_stop” as I feel that is a better representation of a bus stop. One thing that I did not tackle in this project, but feel would have been worthwhile is also changing the “name” tag to “bus_stop_number” so that the node would look like this instead:
<tag k="bus_stop_number" v="2210" />
<tag k="amenity" v="bus_stop" />
However, this change would be considered out of scope in difficulty for the purposes of this project, so I left it as is.

Cities:

When looking through the values that certain tags held, I realized that the “addr:city” tag and “is_in” tag held the same information. There were no nodes that held both an “addr:city” and an “is_in” tag simultaneously. I decided to combine these two tags under the “addr:city” tag.

There were also some city names that required cleaning. For example “Wetaskiwin No. 10, county of”, “Wetaskiwin No. 10, county of, Alberta” and “Wetaskiwin No. 10, county of,

Alberta, Canada" vs "Wetaskiwin County". To clean these up, I used this function, which removes extraneous information like County Number, Province and Country, which are better represented under separate tags.

```
def update_city_name(city):
    # change eg. "No. 7, County of" to "County"
    if "No." in city:
        return city.split("No.")[0] + 'County'
    # if there is a ;, discard string after ;
    if ";" in city:
        city = city.split(";")[0]
    # if there is a comma discard string after comma
    if "," in city:
        city = city.split(',')[0]
    # if there is a #, discard string after #
    if "#" in city:
            city = city.split("#")[0]
    return city
```

Postal Codes:

Lastly, I decided to clean up the "addr:postcode" tag values using this function:

```
def update_postal_code(postcode):
    if len(postcode) < 6:
        return None
    if len(postcode) == 6:
        postcode =  postcode[:3] + " " + postcode[3:]
    if len(postcode) > 7:
        postcode =  postcode[:7]
    #Check that the first two characters are within the possibilities for AB
    if ((postcode[0] == "T") and (int(postcode[1]) in range(5,10)) and
        postcode[2].isalpha() and (postcode[3] == " ") and
        postcode[4].isdigit() and postcode[5].isalpha() and
        postcode[6].isdigit()):
            return postcode
    else:
        return None
```

Edmonton Area Postal Codes are 7 characters long, and follow the pattern T, Digit in range 5 to 9, Letter, Space, Number, Letter, Number. I discarded any values that were under 6 characters long, and discarded extraneous characters from values that were over 7 characters in length. Sometimes the value was 6 characters in length due to a missing space, so I added a space to values with a length of 6 characters. Then I took the postcode and compared it to the pattern. If it still did not follow the exact pattern, it would not be a valid postal code, and therefore I discarded it as a value.

Statistical Overview of the Data:

Please see sql_queries.py for all SQL queries completed as listed below.

File Sizes:

Unfortunately, I was unable to process the entire dataset, as it is 825.5 MB in size, and my computer did not have the capacity. So instead of starting over with a smaller dataset, I decided to complete my processing on only a sample of the data, using sample.py with a k of 15, the sample size that I ended up using was 55.6 MB, or approximately 6.7% of the total database.

edmonton_canada.osm	825.5 MB
sample_k15.osm	55.6 MB
sample_k100.osm	8.3 MB
edmonton_openstreetmaps.db	38 MB
nodes_tags.csv	1.3 MB
nodes.csv	21.7 MB
ways_nodes.csv	6.8 MB
ways_tags.csv	1.8 MB
ways.csv	2 MB

Unique Users, Nodes and Ways

There are 527 Unique Users, 252393 Nodes and 32159 Ways in my sample of the dataset.

Areas Represented:

I realized at this point that it would be easier to create a view of the ways_tags and nodes_tags tables joined together to do queries on, as many of my remaining queries use both.
In my sample of the dataset, 90 areas are represented, the top ten most represented are:

Edmonton	5617 Entries
Strathcona County	577 Entries
Parkland County	490 Entries
Sturgeon County	418 Entries
St. Albert	410 Entries
Leduc County	314 Entries
Westlock County	223 Entries
Spruce Grove	189 Entries
Leduc	175 Entries
Fort Saskatchewan	173 Entries

And the communities that only show up once in the database are: Alcomdale, Alexis, Birch Cove, Crystal Springs, Egremont, Gwynne, Pickardville, Pine Sands, Riviere Qui Barre, Stony

Plain, Tofield, Villeneuve, Wabamun and edmonton. "edmonton" is obviously a typo that was not corrected during my data wrangling phase, and that would be an area for improvement if I were to go back and iterate on the cleaning of the data.

Amenities:

The most common types of amenities listed were:

bus_stop	451 Entries
parking	79 Entries
school	39 Entries
fast_food	37 Entries
restaurant	34 Entries
place_of_worship	25 Entries
fuel	20 Entries
café	14 Entries
bench	11 Entries
bank	9 Entries

This makes me wonder if because I changed all "highway" tags that referred to a bus stop to amenity tags, if that was the reason why the bus_stop amenity has so many more entries than the others. I also wonder if the number of entries for the other amenities should be higher, but they are just tagged in different ways, therefore not showing up on this query. To investigate this, I would need to look through a lot more of the different types of tags more thoroughly, and it might need to be done with the "human eye" rather than programmatically.

Restaurants:

To find out the names of the restaurants, I had to complete a self join on the tags view that I had created. The top results for restaurants were:

Subway	5 Entries
Tim Hortons	4 Entries
local_knowledge	4 Entries
A&W	2 Entries
Booster Juice	2 Entries
KFC	2 Entries
Taco Time	2 Entries

This is fairly representative of the restaurants found locally in the Edmonton area, with 2 notable exceptions: I'm not sure what "local_knowledge" is, but it's not a franchise restaurant, so I'm not sure why it is showing up here. Also I am surprised that McDonalds didn't show up on this list, as it is fairly ubiquitous in the Edmonton area. I am also surprised that there weren't more entries in total, which shows that this database is probably not quite complete, or, and

this is probably more likely, restaurants are being marked under different tags and therefore are not showing up on this search.

Additional Suggestions for improving the Dataset:

1. In some fields, it may improve the data by using drop-down menus instead of a free form field in order to limit input to certain fields.
    - Benefits:
        - This would solve the problem of multiple tag names being used for the same purpose (eg. “addr:street” and “name” or “addr:city” and “is_in”).
        - The amenities tag would also benefit from this because if there is only one place to categorize an amenity, all of the amenities would be in the same place.
    - Anticipated Issues:
        - It is possible that this restriction on inputs would lead to some nodes not being tagged, as they would not fit nicely into the provided input tags.
        - Drop down menus can be harder to navigate for users, and may discourage users from putting in data.
    - Investigation:
        - In this blog, it talks about whether or not websites should use drop-down menus, stating that they can be user-friendly, but they need to be implemented correctly.

Additional Resources:

Stack Overflow
Udacity
Project Example