Toronto Open Street Maps Data

For this project I will be analyzing the Open Street Maps data pertaining to the city of Toronto in Canada as it my city of residence.

- https://www.openstreetmap.org/relation/324211 (https://www.openstreetmap.org/relation/324211)
- http://overpass-api.de/api/map?bbox=-79.6694,43.5635,-79.0851,43.8736 (http://overpass-api.de/api/map?bbox=-79.6694,43.5635,-79.0851,43.8736)

The original OSM XML file was exported from Overpass API using the following coordinates as the bounding area: -79.6694 to -79.0891 and 43.5635 to 43.8736.

The file was 495MB in size, as seen by the following terminal output:

```
ls -hl toronto_map
-rw-r--r--@ 1 NabeelaMerchant  staff   495M 22 Aug 20:33 toronto_map
```

Data Wrangling

In order to translate the xml file into a database, I first needed to take a look at the different nodes, node tags, etc. to get a better sense of the data and see if anything needed to be fixed.

To do that, I first created a sample file of the xml data that contained every 100th top level element. This helped me quickly scan through the data to find the most recurring errors. The code for this can be found in the "1. make_sample_file.py" file. Once I had written the scripts and audited the sample file, I transitioned to the original xml file.

Next, I parsed through the xml file to identify the number of nodes, ways, etc. in the file. The code for this can be found in "2. count_tags.py". The output of the code for the original xml file is as follows:

```
{'bounds': 1,
 'member': 111493,
 'meta': 1,
 'nd': 2340335,
 'node': 2014881,
 'note': 1,
 'osm': 1,
 'relation': 5099,
 'tag': 2307930,
 'way': 343018}
```

Auditing the data

I then audited the sample file to search for inconsistencies in the street names, postal codes, and city names. This was an iterative process that involved identifying the different groups of street name endings, types of postal code recordings, and city names, and then creating a list of acceptable types to filter out any inconsistencies.

These lists were then used later on with the original xml file to identify further inconsistencies and determine what changes needed to be made. The code for this can be found in "3. audit_addresses.py".

Some of the errors I noticed were:

- Variations of abbrevations/capitalizations for Street, Drive, Place, East, West, etc. (st., Pl., E., west)
- Spelling mistakes (Terace instead of Terrace)
- Complete addresses instead of just street names. This often included suite or floor numbers as well. (14th Avenue, Markham, Ont.; Lawrence Avenue West, 1st Floor)
- Postal codes with a space missing between the 3rd and 4th character (M9W1J8 vs. M9W 1J8)
- Postal codes with the last 3 characters missing (M9C)
- Inconsistent city or town names (City of Brampton vs Brampton)
- Incorrect city names (Torontoitalian)

Cleaning the data

Once these errors were identified, I used the script "4. update_osm.py" to go through the xml file and make changes to fix the errors, mainly using dictionaries to map the incorrect entities to the correct ones. The corrections were then written to a new xml file. This had to be repeated a few times to catch stacked errors.

I also noticed that the province was listed under the key 'state' or 'province', and changed the keys to all be 'province' as is the terminology in Canada.

- Correcting abbreviations, spelling mistakes, inconsistent city names, and incorrect city names:

In order to correct these issues I used dictionaries to map specific incorrect words to their correct versions. The code snippet below shows the dictionary for the street name corrections followed by the code that would map the incorrect word to the correct one.

```
# Dictionary to map incorrect street endings to their correct
counterparts
```

```
street_mapping =
{"Lan":"Lane","Ave":"Avenue","Ave.":"Avenue",'avenue':'Avenue',"Blvd":"
Boulevard","Blvd.":"Boulevard",'Ct':'Court','Dr':'Drive','dr':'Drive',"
E":"East","E.":"East",'N':'North','Rd':'Road','rd':'Road',"road":"Road"
,'S':'South','St':'Street','St.':'Street','street':'Street','st':'Stree
t','st.':'Street','Trl':'Trail','Terace':'Terrace','W':'West','W.':'Wes
t','west':'West','Pkwy':'Parkway','Pl':'Place'}

# street_name is the current street name that has been flagged as
problematic (e.g. 'Doris Ave')
# street_type is the category of inconsistency that the street name is
associated with (e.g. 'Ave')
# Using the street_type, the dictionary, street_mapping', can be
accessed to find the replacement for the inconsistent street_type (e.g.
'Ave' --> 'Avenue')
street_name =
street_name.replace(street_type,street_mapping[street_type])
```

To fix city names, the following dictionary and code snippet was used:

```
# Dictionary to map incorrect city names to their correct counterparts
city_mapping = {"City of Brampton":"Brampton","City of
Pickering":"Pickering","City of Toronto":"Toronto","City of
Vaughan":"Vaughan","North York, Toronto":"North York","York,
Toronto":"York","toronto":"Toronto","Town of Ajax":"Ajax","Town of
Markham":"Markham","City of Markham":"Markham","City of Vaughan
(Maple)":"Vaughan","Etoicoke":"Etobicoke","Toronto,
ON":"Toronto","Toronto;City of
Toronto":"Toronto","Torontoitalian":"Toronto","Vaughan
(Concord)":"Vaughan", "Vaughan
(Woodbridge)":"Vaughan","Willowdale":"North
York","markham":"Markham","vaughan":"Vaughan"}

# Finds and fixes city names
if tag.attrib['v'] in city_mapping: #if the value exists in the
dictionary
    tag.set('v', city_mapping[tag.attrib['v']]) #replace the value with
its dictionary counterpart
```

- Correcting postal codes with missing spaces:

Postal codes had different types of errors show up. Either the postal code does not have a space between the 3rd and 4th character or the last 3 characters were missing. To address this, a function called audit_postcode was run on all postal codes. For the first issue a space was artificially created, whereas, for the second issue, I replaced the postal codes with 'Wrong Postal Code' to make it easy to identify later so the entry can be removed.

```
# function used to audit and correct postal codes
def audit_postcode(postal_code):
    m = post_type_re.search(postal_code) #searches for a match to the
post_type_re regular expression
    if m: #if a match is found (space)
        new_pc = postal_code
    elif len(postal_code)==5: #if a match is not found (no space)
        new_pc = postal_code[0:3]+' '+postal_code[3:6] # update the
postal code to create a space
    else: #if no space is found AND the length of the postal code is
not 5 characters
        new_pc = 'Wrong Postal Code' #the postal code gets replaced
with an identifier for future editing
    return new_pc
```

- Correcting complete addresses:

In order to correct a complete address and change it to just its street name, mapping could not be used as each case varied. In some cases, the entire address involved suite numbers or city names. In order to tackle these I stacked a series of if statements that used different regular expressions to find common offenders and remove them appropriately. The snippet below shows the elif statements involved in this process.

```
elif re.search(r'#\w*$',street_name): # removes anything with ' #...'
    m = re.search(r'#\w*$',street_name) #searches for anything with '
#...'
    part_to_remove = ' '+m.group() #adds a space to the front of the
part that was found
    street_name = street_name.replace(part_to_remove,'') #replaces the
part to remove with an empty string
elif (', Suite' in street_name) or (', Ste' in street_name): # removes
Suite or Ste numbers
    street_name = street_name.split(',')[0] #split the street_name at
the comma and keep the first element in the list
elif re.search('\s\b*[Uu]nit',street_name): #removes Unit or unit
numbers
    street_name = re.split('\s*[Uu]nit',street_name)[0] #split
street_name at U/unit at keep the first element in the list
elif 'Floor' in street_name: #removes , Floor and Floor numbers
    if len(street_name.split(',')) > 1: #if street_name splits with a
comma
        street_name = street_name.split(',')[0] #keep the first part of
the list
    else:
        street_split = street_name.split(' ') #split the street name by
spaces
        street_split.remove(street_split[-1]) #removes the last element
in the list (Floor)
```

```
        street_split.remove(street_split[-1]) #removes the last element
in the new list (Floor number)
        street_name = ' '.join(street_split) #joins the remaining
elements in the list together with spaces between them
```

Converting XML to CSV

Once I was happy with the updated xml file, I used the script "5. convert_osm_to_csv.py" to convert the updated xml file to individual csv files for each table based on the schema provided. This resulted in 5 separate csv files that I could then import into the database. The files and their respective sizes (in MB) are as follows:

```
ls -hl nodes.csv nodes_tags.csv ways.csv ways_tags.csv ways_nodes.csv
-rw-r--r--@ 1 NabeelaMerchant  staff   157M  1 Sep 17:59 nodes.csv
-rw-r--r--@ 1 NabeelaMerchant  staff    38M  1 Sep 17:59 nodes_tags.csv
-rw-r--r--@ 1 NabeelaMerchant  staff    19M  1 Sep 18:00 ways.csv
-rw-r--r--@ 1 NabeelaMerchant  staff    52M  1 Sep 18:00 ways_nodes.csv
-rw-r--r--@ 1 NabeelaMerchant  staff    41M  1 Sep 18:01 ways_tags.csv
```

Database Creation

To create the database 'Toronto.db', I used sqlite3 in the command line to create the tables according to the schema provided and import the csv files. The commands used are listed in the text file 'sqlite3_commands.txt'.

The size of the database is 281MB as seen below:

```
ls -hl Toronto.db
-rw-r--r--  1 NabeelaMerchant  staff   281M  7 Sep 11:34 Toronto.db
```

Updating the Database

Now that the data was audited and imported, I could query it to find some interesting information about the city of Toronto. But before that, I needed to finish cleaning up the incorrect postal data. The tables 'ways_tags' and 'nodes_tags' contain the key 'postcodes' with the value 'Wrong Postal Code'. Using the following queries I identified the incorrect postal data and removed the values in the table:

```
select * from nodes_tags where key="postcode" and value="Wrong Postal
Code";
delete from nodes_tags where key="postcode" and value="Wrong Postal
Code";
```

```
select * from ways_tags where key="postcode" and value="Wrong Postal
Code";
delete from ways_tags where key="postcode" and value="Wrong Postal
Code";
```

Database Queries

Now that the database had been cleaned up I moved on to querying. First I wanted to verify that the number of nodes and ways aligned with the values obtained from the python script "2. count_tags.py".

- Number of nodes:

```
select count(id) from nodes;
    2014881
```

- Number of ways:

```
select count(id) from ways;
    343018
```

I then wanted to find the number of unique users that had contributed to this portion of the open street map, along with the top 5 contributing users:

- Number of unique users:

```
select count(users) from (select user as users from nodes group by
user) as u;
    1682
```

- Top 5 contributing users:

```
select user,count(user) from nodes group by user order by count(user)
desc limit 5;
    andrewpmk,1371023
    Kevo,151677
    "Victor Bielawski",95708
    Bootprint,57199
    "Mojgan Jadidi",30997
```

I then wanted to dig deeper into the different keys in the nodes_tags table, and identify the top keys and explore those further.

- Top 10 keys in nodes_tags:

```
select key,count(value) from nodes_tags group by key order by
count(value) desc limit 10;
    source,203202
    street,202844
    housenumber,202783
    city,177477
    highway,50203
    province,28275
    country,28046
    name,25176
    amenity,18474
    crossing,12722
```

- Top 10 amenities in Toronto:

```
select value, count(value) from nodes_tags where key='amenity' group by
value order by count(value) desc limit 10;
    restaurant,2072
    fast_food,2010
    bench,1612
    post_box,1389
    cafe,1073
    parking,947
    waste_basket,867
    bank,706
    pharmacy,498
    telephone,494
```

- Top 5 most popular restaurants:

```
create view rest_id as select id from nodes_tags as rest_id where
value='restaurant';

select nodes_tags.value,count(nodes_tags.value) from rest_id,nodes_tags
where rest_id.id=nodes_tags.id and nodes_tags.key='name' group by
nodes_tags.value order by count(nodes_tags.value) desc limit 5;
    "Swiss Chalet",34
    "Boston Pizza",13
    Eggsmart,12
    "Sunset Grill",11
    "Wild Wing",10
```

- Top 5 most popular cuisines:

```
create view cuisine_id as select id from nodes_tags as cuisine_id where
key='cuisine';

select nodes_tags.value,count(nodes_tags.value) from
nodes_tags,cuisine_id where nodes_tags.id=cuisine_id.id and key =
'cuisine' group by nodes_tags.value order by count(nodes_tags.value)
desc limit 5;
    coffee_shop,599
    pizza,369
    sandwich,333
    burger,205
    chinese,157
```

From the queries above we can see that coffee shops top the list of cuisines in Toronto. Given that Tim Hortons is known as Canada's go-to coffee shop, whereas in America, Starbucks is infamously known to be on every city corner, I wanted to dig into the actual popularity of different coffee shops in Toronto. As coffee shops are listed as restaurants but also under cafes, I need to explore both types of amenities to verify their populatiry.

- Top 5 most popular coffee shops (restaurants):

```
create view coffee_id as select id from nodes_tags as coffee_id where
value="coffee_shop";

select nodes_tags.value,count(nodes_tags.value) from
coffee_id,nodes_tags where coffee_id.id=nodes_tags.id and
nodes_tags.key='name' group by nodes_tags.value order by
count(nodes_tags.value) desc limit 5;
    "Tim Hortons",235
    "Starbucks Coffee",170
    "Second Cup",71
    "Country Style",16
    "Coffee Time",8
```

- Top 5 most popular coffee shops (cafes):

```
create view cafe_id as select id from nodes_tags as cafe_id where
value='cafe';

select nodes_tags.value,count(nodes_tags.value) from cafe_id,nodes_tags
where cafe_id.id=nodes_tags.id and nodes_tags.key='name' group by
nodes_tags.value order by count(nodes_tags.value) desc limi
    "Tim Hortons",276
    "Starbucks Coffee",187
    "Second Cup",76
    "Coffee Time",48
    "Country Style",25
```

In both cases, there are more Tim Hortons outlets than Starbucks in Toronto.

Other ideas about the dataset

In retrospect, having wrangled the xml data and converted in into a database to query, there are a few things I realised along the way and would have changed. While I included a snippet in my python code to change province information (state --> province and all values --> ON), not all the keys and values got caught and I realised that a few values were missed upon querying the database. I proceeded to fix the values after that (shown below), but would try to have a more robust systematic cleanse of the data in the python code itself.

The queries below show the variations in province names in the nodes_tags table and the process of updating them to 'ON'.

```
select value,count(value) from nodes_tags where key='province' group by
value;
    ON,28255
    On,5
    Onatrio,5
    Ont,1
    Ontario,2
    Onterio,1
    on,2
    ontario,4

update nodes_tags set value='ON' where key='province' and value<>'ON';

select value,count(value) from nodes_tags where key='province' group by
value;
    ON,28275
```

When updating the province data for the 'ways_tags' table, I noticed that Florida was listed as a province. Upon further exploring the information associated with that ID I realised that the entry was for an automotive shop in Florida, USA and removed all the information associated with that ID as well. The queries for this, along with those used to update the remaining province variations are shown below:

```
select value,count(value) from ways_tags where key='province' group by
value;
    Florida,1
    ON,9929
    On,5
    Ontario,9
    Ontatio,1
    on,2
    ontario,1

select * from ways_tags where key='province' and value="Florida" group
by value;
    46583206,province,Florida,addr

select * from ways_tags where id="46583206";
    46583206,phone,"(800) 999-5880",regular
    46583206,housenumber,10600,addr
    46583206,opening_hours,"24/7 Available",regular
    46583206,postcode,"339 13",addr
    46583206,name,"Auto Glass America - Fort Myers",regular
    46583206,street,"Colonial Boulevard",addr
    46583206,province,Florida,addr
    46583206,city,"Fort Myers",addr
    46583206,description,"We provide an extensive list of windshield
and auto glass replacement services throughout
    Tampa, Florida.",regular
    46583206,building,automotive,regular
    46583206,shop,car,regular
    46583206,website,http://www.auto-
glassamerica.com/services/windshield-replacement-fort-myers,regular
    46583206,service,repair,regular

select value,count(value) from ways_tags where key='province' group by
value;
    ON,9929
    On,5
    Ontario,9
    Ontatio,1
    on,2
    ontario,1

update ways_tags set value='ON' where key='province' and value<>'ON';

select value,count(value) from ways_tags where key='province' group by
value;
    ON,9947
```

The following queries were used to change the key from 'state' to 'province'. No 'state' keys existed in the 'ways_tags' table so only the 'nodes_tags' table needed to be updated.

```
select value,count(value) from ways_tags where key='state' group by
value;

select value,count(value) from nodes_tags where key='state' group by
value;
    ON,4
    Ontario,5

update nodes_tags set key='province' where key='state';
```

To make the wrangling process for the province data even more robust, I would need to include the province key in the audit of the xml file. This would prevent overwriting incorrect data, such as the Florida entry. However, without writing code to specifically address the incorrect ids and remove them, I would still need to query the database for quick removal as shown above.