Calgary Coffee Shops

By: Nick Shaw

Date: 2016-07-13

Project: P3 from the Udacity Data Analyst Nano Degree (https://www.udacity.com/course/data-analyst-nanodegree--nd002)

Introduction

This report details the investigation of coffee shops in Calgary Canada using data from OpenStreetMap.org (https://www.openstreetmap.org/).

Specifically, the following questions are answered:

1. What does the map data look like? Size of file, number of elements, number of nodes, number of ways and number of unique users.
2. Where does coffee shop rank as an cuisine?
3. What users have added the most kind of coffee shops?
4. What are the most common coffee shop chains?
5. What are the names, phone numbers and postal codes of all the coffee shops.

The investigation will be conducted using python and mongodb. For the python, not all code will be shown in this notebook and can be found in the file osmToMongo.py in the project folder.

The project folder is hosted on github here (https://github.com/shawger/uc-dand).

Source Data

The data was downloaded with a pre-made xml osm file from MapZen. Here (https://s3.amazonaws.com/metro-extracts.mapzen.com/calgary_canada.osm.bz2) is the link.

In the project folder, the file is calgary_canada.osm.

For testing, a sample of calgary_canada.osm was made using createTestFile.py, which was provided as part of this project.

The resulting file is calgary_canada_sample.osm

In [1]:
```
#Creates and uses sample file if True
USE_SAMPLE = False
k = 10

inputFile = "calgary_canada.osm"
sampleFile = "calgary_canada_sample.osm"

if USE_SAMPLE:

    import createTestFile

    createTestFile.createTestFile(inputFile,sampleFile,k)

    print '%s created from %s for testing.' % (sampleFile,inputFile)

    # For the rest of the project the sample file can be used instead of the o
rginal file
    inputFile = sampleFile
```

Load from xml to mongobd

Load the data from xml and convert to json so it can be loaded into mongodb.

osmToMongo.py handles the conversion to json as well as the cleaning of the data.

In [2]:
```
import osmToMongo
from pymongo import MongoClient

data = osmToMongo.process_map(inputFile)
osmToMongo.loadIntoMongoDB(data)
```

```
835823 elelments processed and converted to json.
New file created:               calgary_canada.json
Log file for preproccessing:    pre_process_log.txt
maps document set in mongodb re-loaded
```

While converting the data, 2 kinds of cleaning will happen.

1. Pre-processing: This happens before data is converted to json. Mainly, it is making consistent and valid keys.
2. Post-processing: Once an element is converted to JSON, it will be cleaned according to different rules.

Pre-processing Cleaning

To start, the following rules were applied.

1. 'created' index for elements. Elements have attributes related to their creation ("version", "changeset", "timestamp", "user", "uid"). These are combined into a sub-document with the index 'created'
2. 'pos' index. For elements with lat and lon (latitude and longitude) attributes, they will be combined into an float list (2 elements) and given the index of 'pos'.
3. Valid keys. Elements have sub elements called 'tags'. Each tag has a key (k) and value (v). Before changing to json the keys need to be checked. Keys can only have lowercase letter or an _. For tags with a single ':', a sub-document is created with the string before the ':' being the key in the main document and string after the ':' being the key in the sub-document. Address is a special. There are records with addr:, and instead of using addr for the key, address will be used.

Applying these rules, 15289 tags were dropped (pre_process_log_v0.txt). Here is a sample:

```
Bad tag key, not added: geobase:datasetName
Bad tag key, not added: geobase:acquisitionTechnique
Bad tag key, not added: geobase:datasetName
Bad tag key, not added: geobase:acquisitionTechnique
Bad tag key, not added: catmp-RoadID
Bad tag key, not added: catmp-RoadID
Bad tag key, not added: catmp-RoadID
```

Tags with keys like geobase:acquisitionTechnique and geobase:datasetName were being dropped because they had a capital letter. These appear to be using a naming standard that separates words using capital letters to start words. I added logic to instead use _ to divide these words. For example, datasetName becomes dataset_name.

I added code to replace - with *and to make all words lower cases (after* were added where applicable)

Regex code to find lowerCase letters followed be upper case letters, then mode to make the replacement

```
lowerUpper = re.compile(r'([a-z])([A-Z])')

def _replaceLowerUpper(match):
    lower = str(match.group(1))
    upper = str(match.group(2))
    return lower + "_" + upper.lower()
```

Code inside shape_element (in osmToMongo.py), which converts an xml element to a json document. k is the key for a tag.

```
    k = re.sub(lowerUpper,_replaceLowerUpper,k)
    k = k.lower()
    k = str.replace(k,"-","_")
```

After adding the code, there were 1181 dropped tags due to key name (pre_process_log_v2.txt). Here is a sample:

```
Bad tag key, not added: addr:housenumber:first:right
Bad tag key, not added: name_1
Bad tag key, not added: geobase:route_name1:en
Bad tag key, not added: addr:housenumber:first:right
Bad tag key, not added: addr:housenumber:last:right
Bad tag key, not added: addr:housenumber:first:right
Bad tag key, not added: addr:housenumber:last:right
```

Post-processing Cleaning

I wanted to focus on a few things to help with my questions.

- Make sure things are properly classified as coffee shops.
- Make sure phone numbers are in a standard format
- Make sure postal codes are in a standard format

> Note: For auditing, I added some new fields. In practice, I would take this code (or comment it out) once I was satisfied with the data, as there is no need for it once the data is clean.

For the auditing of the data I will run some queries on the db using the python mongodb driver. Connect to the database with the driver first.

In [3]:
```
client = MongoClient('localhost', 27017)
db = client.p3
```

Coffee Shop Classification

For coffee shops there are different ways of classification. The 'amenity' will be 'cafe' and/or the 'cuisine' will be 'coffee_shop'. Not all are the same. Here is what I mean.

In [4]:
```
cafePipeline = [{"$match" : {"amenity":"cafe"}}]
coffeePipeline = [{"$match" : {"cuisine" :"coffee_shop"}}]
cafeNoCoffeePipeline = [{"$match" : {"$and" : [{"amenity":"cafe"},{"cuisine" :
{"$ne":"coffee_shop"}}] }}]
coffeeNoCafePipeline = [{"$match" : {"$and" : [{"amenity": {"$ne":"cafe"}},{"c
uisine" :"coffee_shop"}] }}]


print "Places with cafe as amenity: %d" % len(list(db.maps.aggregate(cafePipel
ine)))
print "Places with coffee_shop as cuisine: %d" % len(list(db.maps.aggregate(co
ffeePipeline)))
print "Places with cafe as amenity but not coffee_shop as cuisine: %d" % len(l
ist(db.maps.aggregate(cafeNoCoffeePipeline)))
print "Places with coffee_shop as cuisine but not cafe as amenity: %d" % len(l
ist(db.maps.aggregate(coffeeNoCafePipeline)))
```

```
Places with cafe as amenity: 190
Places with coffee_shop as cuisine: 199
Places with cafe as amenity but not coffee_shop as cuisine: 0
Places with coffee_shop as cuisine but not cafe as amenity: 9
```

To start with, running this code on the full dataset I got (the output changed after adding cleaning logic):

```
Places with cafe as amenity: 190
Places with coffee_shop as cuisine: 78
Places with cafe as amenity but not coffee_shop as cuisine: 121
Places with coffee_shop as cuisine but not cafe as amenity: 9
```

To clean this up, I want all coffee shops to look the same. To do this I decided that I would make it so all cafe's have the cuisine coffee_shop and from here on, to use the field, 'cuisine' to to the research.

The follow code is included in the code for cleaning a document:

```
if('amenity' in d and d['amenity'] == 'cafe'):
        d['cuisine'] = 'coffee_shop'
```

After adding this code, the output for the 'Coffee Audit' was:

```
Places with cafe as amenity: 190
Places with coffee_shop as cuisine: 199
Places with cafe as amenity but not coffee_shop as cuisine: 0
Places with coffee_shop as cuisine but not cafe as amenity: 9
```

Now {'cuisine' : 'coffee_shop'} includes all the coffee shops.

Phone Numbers

Check the format of the phone numbers, pick a standard (the most used maybe) then convert all to that standard.

During the clean stage, add a field to all documents that has the format of the number. D is digit Eg (###) ###-####

Code in cleanDocument:

```
if('phone' in d):
        d['phone_format'] = re.sub('[0-9]', '#', d['phone'])
        print d['phone_format']
```

Now run a query to see the common formats.

In [5]:
```
#Query all documents with something in the 'phone_format' field, group by the
 phone format,
#count and sort by the number in each format

formatPipeline = [{"$match" : {"phone_format":{"$exists":{"$ne":"null"}}}},
                  {"$group": { "_id": "$phone_format", "total" : {"$sum":1}}},
                  {"$sort": { "total": -1}}]

results = list(db.maps.aggregate(formatPipeline))

for r in results:

    print '%s : %d' % (r['_id'],r['total'])
```

```
+#-###-###-#### : 277
###-###-#### ext ## : 1
```

From the first run of this, here are the first 5 results:

```
+#-###-###-#### : 135
(###) ###-#### : 47
+# ### ### #### : 25
###-###-#### : 21
(###)###-#### : 10
```

I will use the first format as a template and put the rest into that format.

From my knowledge of Canadian numbers there are 10 digits, not including the country code. If the country code is not included, it is assumed to be 1.

All the digits will be extracted (in order), then put into the correct format.

The code for cleaning is:

```
if('phone' in d and not phoneNumber.match(d['phone'])):

        #The phone number with nothing but digits
        stripped = re.sub('[^0-9]', '', d['phone'])

        #The number should be of length 10 or 11
        #If it is 10, add a 1 to the start
        if(len(stripped) == 10):
            stripped = '1' + stripped

        #Put into the correct format +#-###-###-####
        if(len(stripped) == 11):
            d['phone'] = '+' + stripped[0] \
                         + '-' + stripped[1:4] \
                         + '-' + stripped[4:7] \
                         + '-' + stripped[7:11]
```

and the regex phoneNumber is:

```
phoneNumber = re.compile(r'^[+][0-9][-][0-9]{3}[-][0-9]{3}[-][0-9]{4}$')
```

Now the results of the phone number query are:

```
+#-###-###-#### : 277
###-###-#### ext ## : 1
```

Much better. For more improvement, figure out a format for extensions.

Postal Codes

Check the format of the postal code, and make sure all addresses are using postal codes of the same (most common) format.

In [6]:
```
formatPipeline = [{"$match" : {"address.post_format":{"$exists":
{"$ne":"null"}}}},
                  {"$group": { "_id": "$address.post_format", "total" : {"$su
m":1}}},
                  {"$sort": { "total": -1}}]

results = list(db.maps.aggregate(formatPipeline))

for r in results:

    print '%s : %d' % (r['_id'],r['total'])
```

```
U#U #U# : 1163
```

From the first run:

```
U#U #U# : 1112
U#U#U# : 48
UU U#U #U# : 2
U#U #U#  : 2
U#U-#U# : 1
U#U#U#;U#U #U# : 1
###-###-#### : 1
#### : 1
U#U : 1
```

Not so bad. Some of these just need to be thrown away. The only thing acceptable is (without punctuation or spaces is) L#L#L#. Anything without 3 numbers and 3 letters not in that order is not valid.

The following code was added to clean the postal codes:

```
if ('address' in d \
        and 'postcode' in d['address'] \
        and not postCode.match(d['address']['postcode'])):

        #Remove everything but letter and numbers. Make it upper case
        stripped = re.sub('[^0-9A-Za-z]', '', d['address']['postcode']).upper()

        #Check if the stripped (only letters and numbers) post code is valid
        #Drop it if it isn't
        if(postCodeStripped.match(stripped)):
            d['address']['postcode'] = stripped[0:3] + " " + stripped[3:]
        else:
            d['address'].pop("postcode", None)
```

Using the regex's:

```
postCode = re.compile(r'^[A-Z][0-9][A-Z][\s][0-9][A-Z][0-9]$')

postCodeStripped = re.compile(r'^[A-Z][0-9][A-Z][0-9][A-Z][0-9]$')
```

After the cleaning code was added this is the output of the audit:

```
U#U #U# : 1163
```

Good. That's better.

Analysis

Okay. The data is loaded into the database and cleaned (kind of). Now to run some queries to answer my original questions.

What does the map data look like? Size of file, number of elements, number of nodes, number of ways and number of unique users.

Answered using the following queries:

In [7]:
```
#Use the collstats fucntion to get the file size and nunmber of elemements
stats = db.command("collstats", "maps")

total = stats['count']
size = stats['storageSize']

#Use the count function to get the amount of each type
nodes = db.maps.find({'type':'node'}).count()
ways = db.maps.find({'type':'way'}).count()

#Find the unique users query
#Group by users, then group and count the results.
pipeline = [{"$group": { "_id": "$created.user", "total" : {"$sum":1}}},
            {"$group": { "_id": "null", "total" : {"$sum":1}}}]

results = list(db.maps.aggregate(pipeline))

uniqueUsers = results[0]['total']

print "\nData Summary"
print "================================="
print "Storage Size: %d bytes" % size
print "Total number of elements: %d " % total
print "Total number of nodes: %d " % nodes
print "Total number of ways: %d " % ways
print "Total number of unique users: %d " % uniqueUsers
```

```
Data Summary
=================================
Storage Size: 42860544 bytes
Total number of elements: 835823
Total number of nodes: 745943
Total number of ways: 89838
Total number of unique users: 864
```

Where does coffee shop rank as an cuisine?

Answered using the following query

In [8]:
```
#Query how many non null cuisines sorted
pipeline = [{"$match" : {"cuisine":{"$exists":{"$ne":"null"}}}},
            {"$group": { "_id": "$cuisine", "total" : {"$sum":1}}},
            {"$sort": { "total": -1}}]

results = list(db.maps.aggregate(pipeline))


print '\nTop 10 cuisine types'
i = 0
for result in results:
    if i < 10:
        print '%s:%s' % (result['_id'],result['total'])
    else:
        break
    i += 1
```

```
Top 10 cuisine types
coffee_shop:199
burger:79
pizza:50
chinese:43
sandwich:42
vietnamese:24
japanese:23
american:18
italian:15
mexican:15
```

Coffee shops seem to be the most popular cuisine added.

What users have added the most kind of coffee shops?

Who likes coffee the most? Let's find out!

In [9]:
```
#Find nodes where cuisine is coffee_shop, then group and count the users (wher
e there is a user) then sort
pipeline = [{"$match" : {"cuisine":"coffee_shop"}},
            {"$match" : {"created.user":{"$exists":{"$ne":"null"}}}},
            {"$group": { "_id": "$created.user", "total" : {"$sum":1}}},
            {"$sort": { "total": -1}}]

results = list(db.maps.aggregate(pipeline))

print '\nTop 5 coffee lovers'

i = 0
for result in results:
    if i < 5:
        print '%s:%s' % (result['_id'],result['total'])
    else:
        break
    i += 1
```

```
Top 5 coffee lovers
AKC:65
andrewpmk:30
JamesBadger:10
MuzikMachine:8
abDoug:7
```

AKC seems to like coffee a lot. What else do they like?

In [10]:
```
#Find nodes where user is AKC, then group and count the cuisines (where there
 is a user) then sort
pipeline = [{"$match" : {"created.user":"AKC"}},
            {"$match" : {"cuisine":{"$exists":{"$ne":"null"}}}},
            {"$group": { "_id": "$cuisine", "total" : {"$sum":1}}},
            {"$sort": { "total": -1}}]

results = list(db.maps.aggregate(pipeline))

for result in results:
     print '%s:%s' % (result['_id'],result['total'])
```

```
coffee_shop:65
pizza:24
chinese:23
sandwich:22
vietnamese:12
burger:11
indian:10
japanese:9
mexican:7
asian:7
thai:5
steak_house:5
ice_cream:5
italian:5
chicken:4
greek:3
kebab:3
sushi:2
american:2
seafood:2
pasta:1
authentic;chinese:1
noodle:1
nepalese:1
indian;pakistani:1
japanese;sushi:1
pho:1
korean:1
fish:1
international:1
```

I guess AKC just likes adding things to the openmap. Good for them.

What are the most common coffee shop chains?

Group by the name where cuisine is coffee_shop

In [11]:
```
#Find all coffee shops with names. Group by name and report and sort the count
pipeline = [{"$match" : {"cuisine":"coffee_shop"}},
            {"$match" : {"name":{"$exists":{"$ne":"null"}}}},
            {"$group": { "_id": "$name", "total" : {"$sum":1}}},
            {"$sort": { "total": -1}}]

results = list(db.maps.aggregate(pipeline))
print '\nTop 10 coffee shops'
i = 0
for result in results:
    if i < 10:
        print '%s:%s' % (result['_id'],result['total'])
    else:
        break

    i += 1
```

```
Top 10 coffee shops
Tim Hortons:39
Starbucks:28
Starbucks Coffee:23
Second Cup:11
Jugo Juice:9
Good Earth Cafe:5
Booster Juice:4
Orange Julius:4
Good Earth Coffeehouse and Bakery:4
The Second Cup:2
```

Tim Hortons wins! Wait, there are Starbucks and Starbucks Coffee. They should be the same, and then it would win. In fact there are a few on this list that need cleaning. Good Earth Cafe and Good Earth Coffeehouse and Bakery for example. Cleaning this up would be a future improvement (unless you are a big Timmy's fan).

Probably a good way to attack this would be to create a renaming dict, and use it to clean the names of coffee shops (and other kinds of things). Creating the dict would likely be a manual process...

What are the names, phone numbers and postal codes of all the coffee shops?

In [12]:
```
#Find node where cuisine is coffee_shop and that have name, address and postal
 code
pipeline = [{"$match" : {"cuisine":"coffee_shop"}},
            {"$match" : {"$and": [{"name":{"$exists":{"$ne":"null"}}},
                                  {"phone":{"$exists":{"$ne":"null"}}},
                                  {"address.postcode":{"$exists":
{"$ne":"null"}}}]}}]

results = list(db.maps.aggregate(pipeline))

for r in results:

    print "name: %s, phone: %s, postal code: %s" % (r['name'],r['phone'],r['ad
dress']['postcode'])
```

```
name: Starbuck's, phone: +1-403-208-7100, postal code: T3G 4J8
name: Menchies, phone: +1-403-460-7871, postal code: T2L 2L3
name: Franca's, phone: +1-403-277-0766, postal code: T2E 3P5
name: Tim Hortons, phone: +1-403-255-4886, postal code: T2H 0L3
```

Oh. Thats a little sad. Not heaps of coffee shops with phone numbers and postal codes!

Improvements/Thoughts

- Create 'business name' dict for cleaning business names. This would help to better analyze businesses in the area. The major downside is that it would be difficult to do programatically and would be a long manual process.
- A small amount of users (less then 1000) have added a significant numbers of entries. I wonder why? Is it part of their job?
- Other then coffee, Calgarians really like burgers and pizza. I wonder how this ranks against the rest of North America. My guess is very similar.

References

https://docs.mongodb.com/manual/reference/ (https://docs.mongodb.com/manual/reference/) - Used for information on mongodb

In [ ]: