P3 - DATA WRANGLING PROJECT

OPEN STREETMAP (OSM): SINGAPORE

1. AN OVERVIEW OF SINGAPORE

Singapore is a multi-racial and multi-cultural nation with a population of around 5.6 million, located in Southeast Asia. It is an island city-state, occupying merely 278 square miles of land area, which makes it one of the most densely-populated nation in the world.

I chose Singapore for this project because I grew up there. With my familiarity with this area, I thought the exploration using MongoDB will be very interesting and exciting. I downloaded the OSM file by sending the following web query:
`http://overpass-api.de/api/map?bbox=103.6691,1.2000,104.0817,1.4253`

2. PROBELMS ENCOUNTERED

I noticed several problems with the data regarding street names as I began with sampling the OSM file of Singapore. Before I go on to describe the problems I have encountered, I think it would be appropriate to briefly mention the typical format of a street name in Singapore first.

STREET NAME FORMAT

- *Proper Name*, followed by the ***Street Type*** (e.g. *MacKenzie* ***Road***)
- ***Street Type***, followed by the *Proper Name* (e.g. ***Lorong*** *Ah Soo*)

Numerals *(and occasionally an alphabet)* may follow after the street type, (e.g. *Hougang Street* ***51***). These names indicate that there are more than one branches bearing the same street name.

These two formats display the multi-cultural aspect of Singapore. The first format not only reflects the modern day Singapore as an English-speaking nation, but also its colonial history under the British rule between the years 1819 and 1942. The second format reflects its rich Malayan heritage. The three most common street types associated with the second format from the dataset are: *Jalan* (which translates to 'street' in English), *Lorong* ('lane') and *Taman* ('park'). Others include *Kampong*, *Lengkong* and *Bahru*.

DATA-CLEANING PROBLEMS

The problems I have encountered during the data-cleaning process include:

- Entries inconsistent with the accepted formats
- Problematic mapping text
- Formatting atypical names

Elaborations on these problems follow next.

Inconsistent Formats

There are 39 documents whose street name entries have shown inconsistency from the expected format. They are broadly categorized by:

- The entry is incomplete.
    - (e.g. *Street 81*)

- The entry contains the house or building number.
    - (e.g. ***1013*** *Geylang East Ave 3*, ***Blk 10*** *Ubi Crescent*)

- The entry contains the house unit.
    - (e.g. *East Coast Road* ***#03-09***)

- The entry is not a street name.
    - (e.g. *310074*, *656289*, *Park Regis Singapore*)

I use regular expressions (regex) to screen for inconsistent street names. Below are code snippets to illustrate a couple of regular expressions I use. A complete list of regex I use to screen out inconsistent entries is located in `2_screen.py` under the folder `Python_Codes`.

```
import re

# Inconsistent form #0: Incomplete streetname.
# At least two groups expected: Proper Name, Street Type
ic0 = re.compile("(^|.*\s)[a-zA-Z].*\s[a-zA-Z].*")

# Inconsistent form #1: House units #XX-YY or XX-YY
ic1 = re.compile(".*[0-9]?[#-][0-9].*")
```

Cleaning these problematic entries programatically would be difficult. So I have chosen to, via python scripting, clean these entries "manually" with the following strategy:

1. (Python) Extract the parent xml element whose `addr:street` entry is tested inconsistent. Element is saved to a text file.
   *To make the process programmatically easy, the filenames are uniquely identified by the "tag type" followed by the "element ID". I also create another file named **INDEX** to store those filenames in a list.*
2. (Manual) Update each text file with the correct information.

   - If street name is missing or incomplete, check for clues via the parent element (e.g. if website is available, check for address there, or carry out a web search for address if possible.)
   - Look for the `pos` field value, which contains the latitude and longitude information. Validate that against google maps, which is my gold standard choice.
   - If the location and the address information do not corroborate, mark the problem filename entry in **INDEX** by prefixing it with "***", say.

3. (Python) When the "tag type" and the "element ID" match up with the list of filenames in **INDEX**, the python script will upload the contents of the matching file and replace the parent xml element. If matched to the marked filename, the parent element will be skipped over and not stored.

Following this routine, the element that has the street name entry *Street 81* is dropped because it dose not contain sufficient address information.

Problematic Mapping Text

In this dataset, *St* and *St.* are the only problematic ones. Since *St* and *St.* are also the mapping texts for "Street", it would be disastrous for original entries such as *St Andrew's Road* or *St. George's Lane*. The mapped entries would have been changed to ***Street** Andrew's Road* or ***Street** George's Lane*.

The resolution I have sought for this particular problem is laid out in the **"Saint" streets** segment.

Formatting Atypical Names

I face with additional challenges for the following situation:

- Special capitalization
  - Capital letter after "Mc" or "Mac" (e.g. *McNally Street*)
  - Capital letter after the apostrophe (e.g. *D'Almeida Street*)
  - Capital letter after the dash (e.g. *One-North Gateway*)

- "Saint" streets (e.g. *St. Andrew's Road*, *St. George's Lane*)

Special Capitalization

These are entries that do not follow the typical capitalization rules. Regex is used for these special cases to make sure the capital letters are showing up correctly in the street name entries. The regular expressions can be found in `6_clean_data.py` under the folder `Python_Codes`.

"Saint" Streets

I have chosen to use "St." as the proper form because the non-abbreviated form, "Saint", is somewhat an awkward representation. But as mentioned earlier, "St." is in conflict with the mapping text for "Street".

The solution is simple. A street name entry is first tested via a function whether it is a "Saint" street. If the entry is a "Saint" street, only the text that comes after "St." will be passed on to the mapping function. The replacement of "St." to "Street" is thus avoided. This routine can be found in `6_clean_data.py` under the folder `Python_Codes`.

3. DATA OVERVIEW

This section contains basic statistics about the dataset and the MongoDB queries used to gather them. Here are some information regarding the data files:

Filename	Size	Comments
singapore.osm	145.7 MB	The original OSM file
singapore.xml	147.4 MB	XML format (Cleaned)
singapore.xml.json	159.3 MB	JSON format (MongoDB-ready)

The database is stored to *osm* under the collection *sgp*.

```
$ mongoimport -d osm -c sgp --file sinagpore.xml.json
```

Queries

Number of documents

```
> db.sgp.find().count()
725540
```

Number of nodes

```
> db.sgp.find({"type":"node"}).count()
636204
```

Number of ways

```
> db.sgp.find({"type":"way"}).count()
89300
```

Number of unique users

```
> db.sgp.distinct("created.user").length
1087
```

Top 5 amentities by count

```
> db.sgp.aggregate([{"$match":{"amenity":{"$exists":1}}},
... {"$group":{"_id":"$amenity","count":{"$sum":1}}},
... {"$sort":{"count":-1}},
... {"$limit":5}])
{ "_id" : "parking", "count" : 1623 }
{ "_id" : "restaurant", "count" : 974 }
{ "_id" : "school", "count" : 437 }
{ "_id" : "taxi", "count" : 314 }
{ "_id" : "place_of_worship", "count" : 281 }
```

4. FURTHER EXPLORATION

I want to explore the geospatial feature of MongoDB. The question I have in my mind is this: *What amenities are nearby my former middle school?*

First, I want to inspect the information that the document has on my former middle school, *Victoria School*. I am storing this query result to the variable **vs** so that I can retrieve the data later.

```
> var vs = db.sgp.find({"name":"Victoria School"}).toArray()
> vs[0]
{
    "_id" : ObjectId("58174da74ed0b226097a0215"),
    "name" : "Victoria School",
    "asset_ref" : "93201",
    "created" : {
        "changeset" : "36529447",
        "user" : "JaLooNz",
        "version" : "7",
        "uid" : "741163",
        "timestamp" : "2016-01-12T14:03:11Z"
    },
    "pos" : [
        1.3092614,
        103.9278707
    ],
    "route_ref" : "31;36;43;47;48;55;135;155;196;196e;197;401",
    "location" : "Marine Parade Road",
    "type" : "node",
    "id" : "410467565",
    "highway" : "bus_stop"
}
```

As a side note, I see that this record has neither an `"address"` nor an `"amenity"` field. It has the field `"location"` which could have been an `"address.street"` field instead. Also, I must admit I do not fully understand the purpose of having a `"highway"` field that holds the value of "bus_stop" in this record.

Geospatial Feature

From MongoDB documentation, I gather that I should index the `"pos"` field with the parameter `"2d"`. Then I can use the `"$near"` operator to perform proximity search.

Creating geospatial index on the 'pos' field.

```
> db.sgp.ensureIndex({"pos": "2d"})
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

Below is the query on amenities near my middle school. I am limiting the query to return two results.

```
> db.sgp.find(
...      {"pos": {"$near": vs[0].pos},
...       "amenity": {"$exists": 1}, "address.street":{"$exists": 1}
...      },
...      {"_id":0, "name":1, "pos":1, "amenity": 1, "address.street": 1}
...      ).limit(2).pretty()
{
    "amenity" : "ice_cream",
    "name" : "Ice Cream Chefs",
    "pos" : [
        1.3101987,
        103.9184457
    ],
    "address" : {
        "street" : "East Coast Road"
    }
}
{
    "amenity" : "cafe",
    "name" : "Dutch Colony Coffee Co.",
    "pos" : [
        1.314218,
        103.9193777
    ],
    "address" : {
        "street" : "Frankel Avenue"
    }
}
```

Although I am not familiar with these two establishments, *Ice Cream Chefs* and *Dutch Colony Coffee Co.*, I can vouch that the streets, *East Coast Road* and *Frankel Avenue*, are pretty close to my school along *Marine Parade Road*.

5. ADDITIONAL IDEA TO IMPROVE THE DATA QUALITY

CROWDSOURCING

Crowdsourcing is a powerful way that reaches out to the masses to get external help for "free". It has gained its place in the web service industry over the years. It helps to assume that the dataset I have here is hosted through a web service, although I do not think it is necessary to do so.

By means of gamification, users may participate in adding new entries/documents to the dataset as well as improving data accuracy, consistency and completeness of existing ones. The idea is to use an incentive-driven approach to encourage users to participate in the quality improvement process. So, each time the user does something that improves the dataset, the user gains points. With enough points, the user may earn accolades, and their achievements may get showcased among other ranked users. For paid services, participation may be further tiered using a points-for-rewards approach.

In a crowdsourcing environment, issues such as users entering inaccurate, inconsistent or, in general, poor quality information, is expected to be self-correcting.

Thus, I think the biggest issue is how to ensure success in a crowdsourcing environment. In short, I think it depends on how willing the users are to volunteer and I think the secret ingredient to getting users to volunteer is making the participation fun and useful.

6. CONCLUSION

A NoSQL database, like MongoDB, has a dynamic schema. When I search the dataset for the document about my middle school, its lack of the `"address"` field does not cause the database to break. I enjoy this dynamic schema feature because it means that I am not required to ensure the records/documents conform to a strict, pre-defined schema when creating the database. For me, it means I could avoid the organizational nightmare of creating tables after tables of data in order to build a relational database that works. But as I learn from MongoDB documentation, in the realm of agile development, where the schema of the database changes frequently and fast, a NoSQL database triumphs over relational database in terms of speed and reliability.

REFERENCES

1. Singapore. In *Wikipedia*. Retrieved October 31, 2016, from https://en.wikipedia.org/wiki/Singapore

2. Web Query. In *Overpass-API*. Retrieved September 12, 2016, from http://overpass-api.de/api/map?bbox=103.6691,1.2000,104.0817,1.4253

3. History of the Reuplic of Singapore. In *Wikipedia*. Retrieved October 31, 2016, from https://en.wikipedia.org/wiki/History_of_the_Republic_of_Singapore

4. Create a 2d Index. In *MongoDB*. Retrieved October 28, 2016, from https://docs.mongodb.com/v3.2/tutorial/build-a-2d-index/

5. The Rise of Crowdsourcing (2006, June 1). In *Wired*. Retrieved November 7, 2016, from https://www.wired.com/2006/06/crowds/

6. Gamification. IN *Wikipedia*. Retrieved November 7, 2016, from https://en.wikipedia.org/wiki/Gamification

7. "Ooohh Piece of Candy - Family Guy". *Unknown source*. Retrieved November 7, 2016, from https://s-media-cache-ak0.pinimg.com/236x/42/cd/09/42cd09f10b093e88de8a9aee5f0dbe3f.jpg

8. NoSQL Databases Explained. In *MongoDB*. Retrieved October 29, 2016, from https://www.mongodb.com/nosql-explained

9. Agile Development. In *MongoDB*. Retrieved October 29, 2016, from https://www.mongodb.com/agile-development