






# Project 2 : Data Wrangle OpenStreetMaps Data

Author	Tan Kwok Wee
Email	<a href="mailto:Kwokwee.tan@ge.com">Kwokwee.tan@ge.com</a>

## 1. Python source code for Lesson 6 Quiz

Programing Exercise	Source Code
Iterative Parsing	 lesson6_IterativeParsing.py
Tag Types	 lesson6_TagTypes.py
Exploring Users	 lesson6_ExploringUsers.py
Improving Street Names	 lesson6_ImprovingStreetNames.py
Preparing For Database	 lesson6_preparingForDatabase.py

## 2. OpenStreetMap Sample Project

Map Area : San Francisco, California, United States

<https://www.openstreetmap.org/relation/111968>

<http://metro.teczno.com/#san-francisco>

### 2.1. Introduction

I had chosen to work on the data for San Francisco, California, United States (). I had been to San Francisco for both work and leisure and had many fond memories there. I hope that by working thru the dataset, I would gain more insights about this beautiful city.

I downloaded the osm file from Map Zen : <http://osm-extracted-metros.s3.amazonaws.com/san-francisco.osm.bz2>

I reviewed the data using dataValidation.py and based on the output, I found several issues several issues

- a.) Inconsistent Post Codes ( "CA", "94103-3124", "CA 94080", "944023025")
- b.) Abbreviated Street names ("Hayes St", "California Dr", "San Carlos Ave.")
- c.) Inconsistent State ( "California", "ca")

### 2.2. Problems encountered in the Map

#### 2.2.1. Inconsistent Post Codes

After running the code "dataValidation.py", I can see inconsistent post codes in the data. I intend to standardize the post code data by performing the following cleanup before loading into the database

- a.) Ensuring post codes are 5 digits

Post code data with hyphens will be split and only the first 5 digits of the data retained.

Example : "94103-3124" will be "94103"

- b.) Valid post codes with "CA " appended

Valid post code data will be split and only the 5 digits of the data retained.

Example : "CA 94080" will be "94080"

After loading the data into mongodb, I performed the following query to check

```
Db.mydb.find({'$match' : {'address.postcode' : {'$ne' : {'$regex' : '^\\d{5}$'}}}}}
```

```
Db.mydb.aggregate(
```

```
[{'$match' : { 'address.postcode' : { '$exists' : 1 } }},
```

```
{ '$group' : { "_id" : "$address.postcode", "count" : { '$sum' : 1 } } })
```

### 2.2.2. Abbreviated Street Names

Before loading the data into MongoDB, I also had performed a cleanup of abbreviations that exist in the street names and below shows the dictionary of the abbreviations that being targeted in this exercise.

```
mapping = { "St": "Street",  "St.": "Street",  
            "Ave" : "Avenue",      "Ave." : "Avenue",  
            "Blvd" : "Boulevard",   "Blvd." : "Boulevard",  
            "Rd." : "Road",         "Rd" : "Road"  
            }
```

### 2.2.3. Inconsistent State Names




I also had noticed inconsistent state names in the data and updated all inconsistent state names. If the data contains the inconsistent state names in the below list, I will default them to be "CA"

```
incorrectstate = ["California","ca"]
```

## 3. Data Overview

This section describes an basic overview of the san-francisco.osm dataset loaded into MongoDB and the related MongoDB queries used.

### 3.1. Source Codes

File Description	File Name	Source Code
Program to extract base statistics about the osm file	dataValidation.py	 dataValidation.py
Program to clean the data inconsistencies , transform the data into JSON and load the data into MongoDB	CleanTransformLoadOsmFile.py	 CleanTransformLoadOsmFile.py
Program to extract data from mongoDB and plot a bar chart	Prj2_visualization.py	 prj2_visualization.py

### 3.2. File Size

File Name	Size
san-francisco.osm	318.6 MB
san-francisco.osm.json	465.5 MB

### 3.3. Number of Documents

```
> db.mydb.count()
```

```
1564506
```

### 3.4. Number of Nodes

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

```
1410191
```

### 3.5. Number of Ways

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

```
154315
```

### 3.6. Number of Unique Users

```
> db.mydb.distinct("created.user").length
```

```
1288
```

### 3.7. Top Contributor

```
> db.mydb.aggregate([
  {"$group": { "_id": "$created.user", "count": { "$sum": 1 } }},
  {"$sort": { "count": -1 } }, { "$limit": 1 } ])
```

```
{ "_id": "oldtopos", "count": 334076 }
```

### 3.8. Top 10 Contributing Users

```
> db.mydb.aggregate([
  {"$group": { "_id": "$created.user", "count": { "$sum": 1 } }},
  {"$sort": { "count": -1 } },
  {"$limit": 10 } ])
```

```
{ "_id": "oldtopos", "count": 334076 }
{ "_id": "KindredCoda", "count": 144613 }
{ "_id": "osmmaker", "count": 140043 }
{ "_id": "DanHomerick", "count": 119462 }
{ "_id": "nmixer", "count": 77785 }
{ "_id": "woodpeck_fixbot", "count": 46008 }
{ "_id": "StellanL", "count": 43335 }
{ "_id": "oba510", "count": 38785 }
{ "_id": "dchiles", "count": 38472 }
{ "_id": "Speight", "count": 30346 }
```

### 3.9. Number of Users having 1 post

```
> db.mydb.aggregate([
  {"$group": { "_id": "$created.user", "count": { "$sum": 1 } }},
  {"$match": { "count": 1 } },
  {"$group": { "_id": "$count", "Num_users": { "$sum": 1 } } } ])
```

```
{ "_id": 1, "Num_users": 292 }
```

### 3.10. Top 10 referenced Node

```
> db.mydb.aggregate([ {"$unwind":"$node_refs"},
{"$group": { "_id": "$node_refs", "count": { "$sum": 1 } }},
{"$sort": { "count": -1 }}, {"$limit": 10}])
```

```
{ "_id": "1384557222", "count": 8 }
{ "_id": "1384625712", "count": 8 }
{ "_id": "1384441406", "count": 8 }
{ "_id": "1384459647", "count": 8 }
{ "_id": "1384567839", "count": 8 }
{ "_id": "53070089", "count": 8 }
{ "_id": "1384682756", "count": 8 }
{ "_id": "1384524657", "count": 8 }
{ "_id": "2266373763", "count": 7 }
{ "_id": "2266373764", "count": 7 }
```

### 3.11. Top 10 Postcode by count

```
> db.mydb.aggregate([
{"$match": { "address.postcode": {"$exists": true}}},
{"$group": { "_id": "$address.postcode", "count": { "$sum": 1 } }},
{"$sort": { "count": -1}},
{"$limit": 10}])
```

```
{ "_id": "94063", "count": 358 }
{ "_id": "94587", "count": 250 }
{ "_id": "94109", "count": 200 }
{ "_id": "94103", "count": 192 }
{ "_id": "94061", "count": 161 }
{ "_id": "94114", "count": 129 }
{ "_id": "94113", "count": 111 }
{ "_id": "94110", "count": 65 }
{ "_id": "94107", "count": 63 }
{ "_id": "94102", "count": 63 }
```

## 4. Additional Ideas

### 4.1. Contributor Statistics

I believe that the openstreetmap data can be improved with data from other private/public sites ( yelp, tripadvisor, translink api, bart.gov and etc. ). By having complimentary data like restaurants rating, places of interest, school ranking, availability of public transport etc. with the existing openstreetmap data, the user will be provided with not only location details but reviews/ratings/places of interest/public transport.

I believe by incorporating more data to openstreetmap , we can get more people to participate and contribute. The team can also validate data from various sources to improve data accuracies/consistencies.

I also hope that people can share day tour/running/cycling/swimming routes on the map so that people can gain awareness and lead a healthier lifestyle

### 4.2. Additional data exploration using MongoDB Queries

#### 4.2.1. Top 10 Appearing amenities

```
> db.mydb.aggregate([
{"$match": { "amenity" : { "$exists" : true } }},
{"$group": { "_id" : "$amenity", "count" : { "$sum" : 1 } }},
{"$sort": { "count" : -1 }},
{"$limit" : 10 }])
```

```
{ "_id" : "parking", "count" : 2958 }
{ "_id" : "restaurant", "count" : 1857 }
{ "_id" : "school", "count" : 1338 }
{ "_id" : "place_of_worship", "count" : 1064 }
{ "_id" : "fire_hydrant", "count" : 698 }
{ "_id" : "post_box", "count" : 588 }
{ "_id" : "cafe", "count" : 574 }
{ "_id" : "bench", "count" : 445 }
{ "_id" : "fast_food", "count" : 436 }
{ "_id" : "drinking_water", "count" : 333 }
```

#### 4.2.2. Top 10 religion based counts of place of worship

```
> db.mydb.aggregate([
{"$match": {"amenity": "place_of_worship", "religion": {"$exists": true}}},
{"$group": {"_id": "$religion", "count": {"$sum": 1}} {"$sort": {"count": -1}},
{"$limit": 10}])

{ "_id": "christian", "count": 979 }
{ "_id": "buddhist", "count": 16 }
{ "_id": "jewish", "count": 14 }
{ "_id": "muslim", "count": 4 }
{ "_id": "unitarian", "count": 2 }
{ "_id": "scientologist", "count": 2 }
{ "_id": "unitarian_universalist", "count": 1 }
{ "_id": "shinto", "count": 1 }
```

#### 4.2.3. Top 10 denominations of Christianity based on counts of place of worship

```
> db.mydb.aggregate([
{"$match": :
{"amenity": "place_of_worship", "religion": {"$exists": true},
"religion": "christian",
"denomination": {"$exists": true}}},
{"$group": {"_id": "$denomination", "count": {"$sum": 1}}},
{"$sort": {"count": -1}},
{"$limit": 10}])

{ "_id": "baptist", "count": 200 }
{ "_id": "catholic", "count": 128 }
{ "_id": "methodist", "count": 70 }
{ "_id": "lutheran", "count": 66 }
{ "_id": "presbyterian", "count": 50 }
{ "_id": "episcopal", "count": 42 }
{ "_id": "scientist", "count": 19 }
{ "_id": "mormon", "count": 17 }
{ "_id": "seventh_day_adventist", "count": 15 }
{ "_id": "jehovahs_witness", "count": 14 }
```



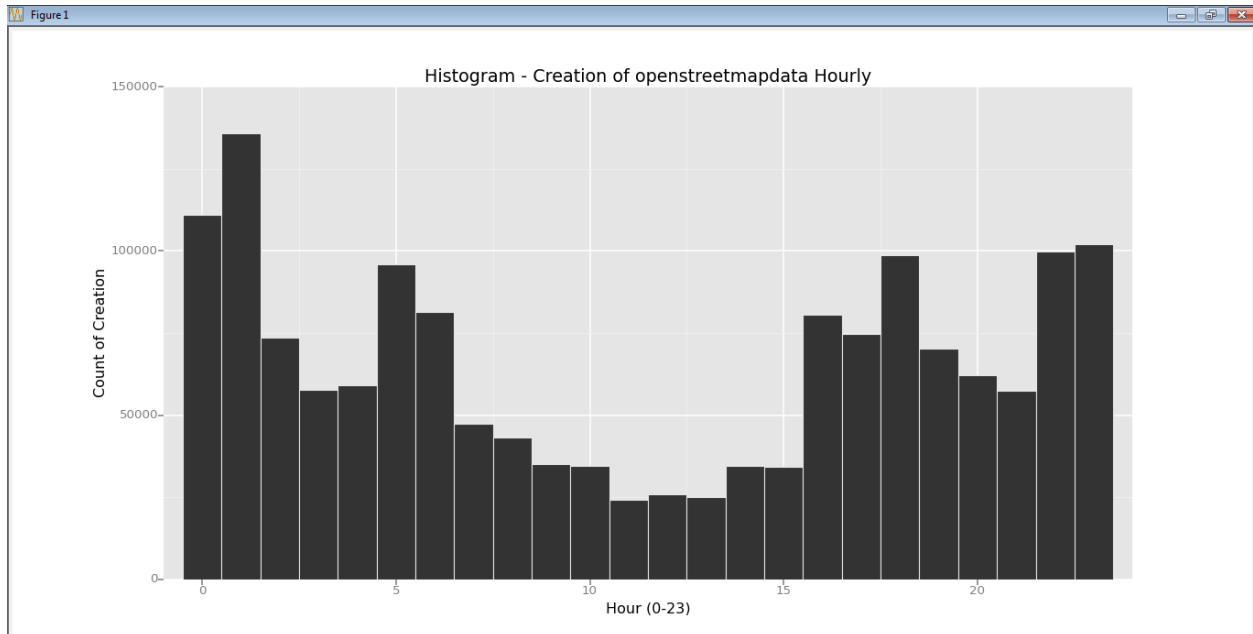
#### 4.2.4. Most Popular Cuisines

```
> db.mydb.aggregate([
{"$match": {"cuisine": {"$exists": true}}}, {"$group": {"_id": "$cuisine", "count": {"$sum": 1 }}},
{"$sort": {"count": -1}},
$limit": 10]])
```

```
{ "_id": "mexican", "count": 163 }
{ "_id": "coffee_shop", "count": 144 }
{ "_id": "burger", "count": 141 }
{ "_id": "pizza", "count": 124 }
{ "_id": "sandwich", "count": 99 }
{ "_id": "italian", "count": 95 }
{ "_id": "chinese", "count": 93 }
{ "_id": "american", "count": 73 }
{ "_id": "thai", "count": 71 }
{ "_id": "japanese", "count": 71 }
```

## 4.2.5. Visualization

I am curious at which time of day were the elements created. From the histogram below we can see that the users were most active during (midnight to 1am)



```
> db.mydb.aggregate([ {"$project" : {"hourOfDay" : {"$hour" : "$created.timestamp"}},  
{"$group":{"_id" : "$hourOfDay", "count" : {"$sum" : 1}}},  
{"$sort" : {"_id" : 1}}])
```

```
{ "_id" : 0, "count" : 111098 }  
{ "_id" : 1, "count" : 135829 }  
{ "_id" : 2, "count" : 73552 }  
{ "_id" : 3, "count" : 57732 }  
{ "_id" : 4, "count" : 59220 }  
{ "_id" : 5, "count" : 96027 }  
{ "_id" : 6, "count" : 81397 }  
{ "_id" : 7, "count" : 47247 }  
{ "_id" : 8, "count" : 43278 }  
{ "_id" : 9, "count" : 34985 }  
{ "_id" : 10, "count" : 34631 }  
{ "_id" : 11, "count" : 24118 }  
{ "_id" : 12, "count" : 25844 }  
{ "_id" : 13, "count" : 25159 }  
{ "_id" : 14, "count" : 34556 }  
{ "_id" : 15, "count" : 34367 }  
{ "_id" : 16, "count" : 80500 }  
{ "_id" : 17, "count" : 74754 }  
{ "_id" : 18, "count" : 98768 }  
{ "_id" : 19, "count" : 70123 }
```

```
{ "_id" : 20, "count" : 62086 }  
{ "_id" : 21, "count" : 57523 }  
{ "_id" : 22, "count" : 99703 }  
{ "_id" : 23, "count" : 102009 }
```

## References

Python - Mongodb <http://api.mongodb.org/python/current/tutorial.html>

Mongodb <http://docs.mongodb.org/>

Python – xml

<https://docs.python.org/2/library/xml.etree.elementtree.html#xml.etree.ElementTree.Element.findall>

ggplot- [http://ggplot.yhathq.com/docs/geom\\_point.html](http://ggplot.yhathq.com/docs/geom_point.html)