# Data Wrangling OpenStreetMap

OpenStreetMap (OSM) is a collaborative project to create a free editable map of the world.Map data is collected from scratch by volunteers performing systematic ground surveys using tools such as a handheld GPS unit, a notebook, digital camera, or a voice recorder. The data is then entered into the OpenStreetMap database. Since the data can be entered by anyone manually or any other means they are prone to errors. This data analysis involves exporting osm file(San Francisco,California)and identifying errors,fixing it and loading it into sql database and exploring the data.In this report we have the following things discussed.

## 1) Problems encountered in the osm data.

### a) Inconsistent and overly- abbreviated street names.

The street names are audited and we can see that there are some inconsistency in the street names like “avenue” for “Avenue”.We also have the over abbreviated street names like “Plz” for Plaza and “St.” for Street.So we map the street names to consistent ones.

Santa Cruz avenue=>Santa Cruz Avenue
Woodside Plz => Woodside Plaza
Webster St. => Webster Street

### b) Inconsistent postal codes.

We have some inconsistent post codes like
CA 94607
94118-4504

These were converted to normal five digit form zipcodes.Like “CA 94067” to 94067 and 94118-4504 to 94118.

## 2) Data overview.

By using iterative parsing to process the map file we find out not only what tags are there, but also how many, to get the feeling on how much of which data you can expect to have in the map.

```
'bounds': 1,
 'member': 47666,
 'nd': 5372072,
 'node': 4517533,
 'osm': 1,
 'relation': 4722,
 'tag': 1625808,
 'way': 517238
```

### a) File sizes.

San-francisco_california.osm 967.MB

Ways.csv 30.8 MB

Nodes.csv 380 MB

Ways_tags 48 MB

Ways_nodes 128.9MB

Nodes_tags 8.6 MB

Final_osm.db 511.5 MB

### b) Number of nodes.

```
sqlite> SELECT COUNT(*) FROM nodes;

4517533
```

### c) Number of ways.

```
sqlite>  SELECT COUNT(*) FROM ways;

517238
```

**d) Number of unique Users**

```
sqlite>  SELECT COUNT(DISTINCT(u.uid))
   ...> FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) u;
```

```
2308
```

By using iterative parsing to process the map file we find out not only what tags are there, but also how many, to get the feeling on how much of which data you can expect to have in the map.

```
'bounds': 1,
 'member': 47666,
 'nd': 5372072,
 'node': 4517533,
 'osm': 1,
 'relation': 4722,
 'tag': 1625808,
 'way': 517238
```

**e) Top 10 contributing users**

```
sqlite> sqlite> SELECT e.user, COUNT(*) as num

   ...> FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e

   ...> GROUP BY e.user

   ...> ORDER BY num DESC

   ...> LIMIT 10;
```

```
ediyes,941356

Luis36995,719336

Rub21,408692

RichRico,224605

calfarome,185295

oldtopos,167916

KindredCoda,155454
```

```
karitotp,135173

samely,125908

abel801,108423
```

## 3) Additional Data Exploration and Cleaning

**Number of Postal operators;**

```
select count(*), value from nodes_tags where key='operator' and id in (select id from nodes_tags where key='amenity' and value in ('post_office','post_box')) group by value;
```

```
12,FedEx
1,Fedex
2,"The Postal Chase"
12,UPS
1,"UPS; DHL; FedEx"
1,"US Government"
16,"US Mail Service"
2,"US Postal Service"
342,USPS
2,"United Parcel Service"
1,"United State Postal Service"
7,"United States Postal Service"
1,ups;fedex
1,usps
```

While working with sample osm there were only postal operators and just one entry had an error.But with large data it has so many operators and some are inconsistent.So updating the data for consistency so while querying for FedEx there will be 13 count instead of 12.Since most people are entering “FedEx” lets keep it.

```
sqlite>update  nodes_tags set value=’FedEx’ where key='operator' and value=’Fedex’;
```

```
sqlite> update nodes_tags
   ...> set value='USPS'
   ...> where key='operator'
   ...> and
   ...> value in ('United State Postal Service','United States Postal Service','US Mail Service','US Postal Service','usps');
```

```
sqlite>update  nodes_tags set value='UPS' where key='operator' and value in('United Parcel Service','ups');
```

Lets check now,

```
sqlite>Select value, count(*)from nodes_tags where key='operator' and id in (select id from nodes_tags where key='amenity' and value in ('post_office','post_box')) group by value;

FedEx,13
"The Postal Chase",2
UPS,14
"UPS; DHL; FedEx",1
"US Government",1
USPS,369
Ups;fedex,1
```

**Wheel chair accessibility details;**

```
sqlite> select value,count(*) from nodes_tags where key='wheelchair' group by value;

limited,57
no,90
private,3
unknown,1
yes,870
```

**Capacity of the bicycle Parking**

```
Select value as capacity, count(*) from nodes_tags where key='capacity' and id in (select id from nodes_tags where value='bicycle_parking'and key='amenity' ) group by value
   ...> order by count(*) desc;


capacity    count(*)
----------  ----------
2          84
4          58
8          27
6          25
10          15
3          11
20          9
12          8
```

```
30          5
16          4
14          3
18          3
9           2
22          1
24          1
268         1
28          1
32          1
350         1
40          1
44          1
5           1
50          1
7           1
```

## Improving and analysing data.

### Comparing OSM data with authoritative resources, and rule-based (manual or automatic) self-detection.

From the dataset provided in the openstreetmap ,one difficulty would be dealing with naming inconsistencies between names eg. USPS and usps and some zipcode errors like just typing CA or too many or less digits.

If OSM can use some pattern based learning like most people use USPS for the post box operator.So OSM can capture such patterns and correct the bugs.Though this could be overcome with careful string handling and a human verifying inputted data.

Some auto validation while entering data can be done  to the page data is being entered. For eg if state is in USA then the zipcode validation can be turned to just digits and number of digits limited to 5.May be that would render a large amount of clean data into OSM.

Answers for the questions asked by the reviewer.

You mentioned that comparison of OSM data with external sources and auto correction/validation could improve the quality of your dataset. Indeed, both ideas sound great. Please note that you are also expected to give a 'thoughtful discussion about the benefits as well as some anticipated problems in implementing the improvement.' The benefits of both ideas are quite clear, but I think that your report would benefit from a more detailed discussion of the potential problems/challenges associated with

implementations of the proposed improvements. For example, you mentioned that 'one difficulty would be dealing with
naming inconsistencies'. Nevertheless, this difficulty is central to the actual goal of the auto correction feature to be implemented. Can you think of additional challenges associated with this implementation? Similarly, it would be important to mention some challenges associated with the validation of OSM with external sources. For example, how would you retrieve such data? Would you use an API? Would you be using a free service? Would there be any contraints on the number of calls? Would there be any data licensing/property issues?

==========================================================================

- Comparison of OSM data with external sources
  - Database might not be fully open. Some of them might be paid and some have restrictive licenses on how they can be used.. For example, they should not be used in any commercial manner.
  - Various data formats like Tiger, XML, RDBMS, Hadoop, REST api etc.. pose a challenge of writing different adapters to fetch data.
  - Adapters also need to be written to convert each to a OSM compatible format for comparison.
  - Third party resource constraints/quota might prevent us from retrieving all the data. Speed might also be throttled.
  - The schema for data may not be available in certain cases.
  - There might be inconsistencies between certain providers. For example, Google maps might be updated later compared to tiger db, and so there will be differences. We need to compare freshness of data to solve these.
  - Client side resources to make all the remote calls to fetch data and compare them. It might be infeasible for example to fetch all data from different sources at once. We might have to partition the data by location and fetch all sources for a small location at a time.

- Auto correction / validation
  - Needs to be aware of country, state and other regional level quirks. So different rules might need to be written based on lat/long.
  - Needs to run fast enough for users not to feel burdened by additional latency of checking.
  - Could use machine-learning which is trained by a smaller sample of human experts who could manual clean and input the data. They could feed both original (negative) examples and the cleaned (positive) examples.
  - Showcase auto corrected entries to users before finalizing, and let them provide feedback. Lot of rules for auto correction or validation could also be crowd-sourced by users who enter them.
  - Validation could be done at client itself (javascript) or on the server side (c++ or java). The former has the advantage that it could be quicker, but consumes user cpu. So if user has an old device, it will run slow. The latter

has the advantage of offloading cpu to server, but could feel slow if network latency is high.

## Conclusion

The San Francisco,california OSM dataset is  quite large and messy. While it is clear that the data is not 100% clean, I used python code to clean some data and some sql queries to further clean it.

http://www.openstreetmap.org/export#map=11/41.4980/-81.7064
https://mapzen.com/data/metro-extracts/metro/san-francisco_california/