**Introduction:**

In this project we are going to get the street map data for Seattle from OpenstreetMap. We are going to clean this data and load it into sql tables and then do some exploratory data analysis on that.

**Location:**

Seattle, WA USA.

(<https://mapzen.com/data/metro-extracts/metro/seattle_washington/>)

**Initial Analysis:**

Using the sample file generation code attached in sample.py file we got a sample file for the Seattle xml data we received from OpenStreetMap. From the initial exploration here are the details of total tags we received.

'node': 800472, 'nd': 892450, 'member': 8845, 'tag': 447550, 'relation': 1026, 'way': 79502, 'osm': 1

I also checked for what kind of tags we have in this xml. I have divided them in 4 categories.

1. "lower", for tags that contain only lowercase letters and are valid,
2. "lower\_colon", for otherwise valid tags with a colon in their names,
3. "problemchars", for tags with problematic characters, and
4. "other", for other tags that do not fall into the other three categories.

Here are the results for the sample file.

{'lower': 235754, 'lower\_colon': 204832, 'other': 6964, 'problemchars': 0}

**Problems in Street Map data:**

1. **Inconsistent Street Names:**

First problem that I see after initial exploration of data is that street names are not consistent across the xml file. For e.g. Avenue is written in few different ways in the file. In some tags it is written as ‘ave.’, in some it is written as ‘ave’ or ‘Ave.’. So we have make this uniform. Another issue I can see with Seattle data is inconsistency with the region in address. To specify region NorthEast few tags have value ‘NE’ or ‘northeast’ or ‘NorthEast’. Similar examples can be seen for regions South, North, SouthWest.

I have written functions to fix this issue. This can be seen in file CorrectStreetNames.py

* audit\_street\_type function search the input string for the regex. If there is a match and it is not within the "expected" list, add the match as a key and add the string to the set.
* is\_street\_name function looks at the attribute k if k="addr:street"
* audit function will return the list that match previous two functions. After that, we would do a pretty print the output of the audit. With the list of all the abbreviated street types we can understand and fill-up our "mapping" dictionary as a preparation to convert these street name into proper form.

Here are few examples:

1. **NorthEast**

100th Avenue Northeast => 100th Avenue NorthEast

157th Place NE => 157th Place NorthEast

1. **Avenue:**

6th Ave => 6th Avenue

south puget sound ave => south puget sound Avenue

More details for this can be found in file Corrected\_Street\_Names.txt.

1. **Inconsistent Post Codes:**

To fix the post code issue with the data we can use a function similar to what I wrote for Street Name. There are few cases where the post codes are not correct. I have written functions to fix this issue. This can be seen in file CorrectPostCodes.py

* is\_valid\_postcode function looks at the attribute k if k=" addr:postcode"
* audit\_postcode function looks at the first two digits of the value to see if they are numbers and if they are 98 as zip code in seattle start with 98.

Here are few examples:

1. Olympia, 98502 => 98502
2. Lacey, WA 98503 => 98503
3. V9A3N8 => None

More details for this can be found in file Corrected\_Post\_Codes.txt.

**Extract Data in CSV:**

I have written function shape\_element to get the tags in various csv files for to load in sqlite as tables.

Shape Element Function returns a dictionary based on conditions. This can be seen in file export\_csv.py

**If the element top level tag is "node"** it will return a dictionary in format : {"node": .., "node\_tags": ...}

The "node" field holds a dictionary of the following top level node attributes:

* + id
  + user
  + uid
  + version
  + lat
  + lon
  + timestamp
  + changeset

All other attributes have been ignored

The "node\_tags" field holds a list of dictionaries, one per secondary tag. Secondary tags are

child tags of node which have the tag name/type: "tag". Each dictionary have the following

fields from the secondary tag attributes:

* + id: the top level node id attribute value
  + key: the full tag "k" attribute value if no colon is present or the characters after the colon if one is.
  + value: the tag "v" attribute value
  + type: either the characters before the colon in the tag "k" value or "regular" if a colon is not present.

Additionally,

- if the tag "k" value contains problematic characters, the tag has been ignored

- if the tag "k" value contains a ":" the characters before the ":" has been set as the tag type

and characters after the ":" has been set as the tag key

- if there are additional ":" in the "k" value they and they have been ignored and kept as part of

the tag key. For example:

<tag k="addr:street:name" v="Lincoln"/>

Is turned into:

{'id': 12345, 'key': 'street:name', 'value': 'Lincoln', 'type': 'addr'}

- If a node has no secondary tags then the "node\_tags" field contains an empty list.

**If the element top level tag is "way"** it will return a dictionary in the format {"way": ..., "way\_tags": ..., "way\_nodes": ...}

The "way" field holds a dictionary of the following top level way attributes:

* + id
  + user
  + uid
  + version
  + timestamp
  + changeset

All other attributes have been ignored

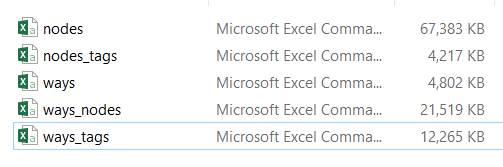
The "way\_tags" field holds a list of dictionaries, following the exact same rules as for "node\_tags".

Additionally, the dictionary has a field "way\_nodes". "way\_nodes". This holds a list of dictionaries, one for each ‘nd’ child tag. Each dictionary should have the fields:

* + id: the top level element (way) id
  + node\_id: the ref attribute value of the nd tag
  + position: the index starting at 0 of the nd tag i.e. what order the nd tag appears within the way element

**File Size Details:**

Here are file size details for all 5 files.



**Load CSV files into Sqlite:**

To load CSV files into Sqlite as tables following statements are run on sqlite window.

1. Sqlite3 Seattle.db – This one creates the DB
2. .mode csv – Enables the mode to load csv file.
3. import nodes.csv nodes;
4. import nodes\_tags.csv nodes\_tags;
5. import ways.csv ways;
6. import ways\_nodes.csv ways\_nodes;
7. import ways\_tags.csv ways\_tags;

**Data Analysis Using Sqlite:**

1. Get Number of Nodes:

Select count(\*) from nodes;

800472

1. Get Number of Ways:

Select count(\*) from ways;

79502

1. Number of Unique Users:

SELECT COUNT(DISTINCT(e.uid))

FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;

2397

1. Top 10 Contributing Users:

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;

|  |  |
| --- | --- |
| Glassman | 130560 |
| SeattleImport | 73579 |
| tylerritchie | 65548 |
| woodpeck\_fixbot | 56652 |
| alester | 36628 |
| Omnific | 36572 |
| Glassman\_Import | 22588 |
| CarniLvr79 | 21148 |
| STBrenden | 20273 |
| Brad Meteor | 19319 |

1. Number of SuperMarkets in Seattle:

SELECT nodes\_tags.value, COUNT(\*) as num

FROM nodes\_tags

JOIN (SELECT DISTINCT(id) FROM nodes\_tags WHERE value='supermarket') i

ON nodes\_tags.id=i.id

WHERE nodes\_tags.key='shop'

GROUP BY nodes\_tags.value

ORDER BY num DESC;

* + 26

1. Top 10 Amenities:

SELECT value, COUNT(\*) as num

FROM nodes\_tags

WHERE key='amenity'

GROUP BY value

ORDER BY num DESC

LIMIT 10;

|  |  |
| --- | --- |
| **bench** | **357** |
| **bicycle\_parking** | **334** |
| **restaurant** | **312** |
| **cafe** | **158** |
| **waste\_basket** | **141** |
| **fast\_food** | **115** |
| **school** | **93** |
| **place\_of\_worship** | **86** |
| **toilets** | **80** |
| **parking** | **72** |

**Conclusion:**

**Ideas for Improvement:**

After reviewing the data I feel we have few issues as more than thousand users have contributed to this data.

1. I would recommend to create a documentation to provide users sample details about the format to put data.
2. If possible we should verify this data with other data sources such Google and Bing.

To further explain this point we can look at the node ‘59646398’ in seattle osm file.

From the postcode mentioned in the data we come to know that something is not adding up correctly. If we cross reference the lat (48.4682425) and lon (-123.3361108) values we get a location in Victoria Canada. Hence if we verify the data using lat and lon values with other data sources we can move these nodes to the correct bucket.

Overall I feel with the data processing we did as part of this project will clean the data. I think with more robust data processor it is possible to provide better data for OpenStreetmap.org.

**Issues with the Solutions:**

1. One of the problem with the documentation solution is it is very difficult to cover all the cases in documentation and whatever is missed then that comes to the individual user about how he understands the document and the data he has.
2. The cross reference of the data with Google or Bing may not be a cost effective solution.

Also for this to implement on such grand scale we may need a big developers team from various places in the world to understand some of the data cleaning issues. As I have lived in Seattle I know that SE in address means South East but a person from some other part of the world may not understand this.

**References:**

1. Udacity DataWrangling Course.

2. <http://www.sqlitetutorial.net/>