Name: Navina Govindaraj
Date: July 2017

Project Overview

- Choose any area of the world from https://www.openstreetmap.org
- Use the downloaded XML data to perform data munging techniques to clean and assess data quality for validity, accuracy, completeness, consistency and uniformity
- Parse the elements in the OSM XML file, transforming them from document format to tabular format, making it possible to write to .csv files.
- Import csv files to SQL database as tables and perform analysis

Map Area

- Being a Bay Area resident, I chose to explore the city and county of San Francisco.

Location	San Francisco, CA, United States
Map URL	https://mapzen.com/data/metro-extracts/metro/san-francisco_california/
Map Position	https://www.openstreetmap.org/relation/111968
Coordinates	https://www.openstreetmap.org/export#map=10/37.7852/-122.7278

Data Wrangling - Problems Encountered in the Map

Note: A sample of the original data set was used for testing purposes and to get an initial idea of the dataset. Once it was confirmed that the schema of the transformed .csv files matched the schema of the original XML OSM dataset, analysis has been performed on the complete dataset.

- Through data auditing, it was identified that the dataset contains information in the form of tags, with 'k' indicating the attribute name and the 'v' indicating values of the attribute.
- From the San Francisco Open Street Map dataset, the following problems were identified.

1. Auditing City Names (attribute: 'addr:city')

- The state name has been included as a suffix to the city name in some cases
- Irrelevant numbers have been added instead of a city name
- To clean the data, the inconsistent capitalization has been corrected with a standard formatting, the state name attached as a suffix to the city names has been removed, and fields with irrelevant numbers have been removed.
  file reference: audit.py*

2. Auditing Country Names (attribute: 'addr:country')

- The country name has several variations such as USA, US, United States and misspelled abbreviations such as UA.
- Since the most common abbreviation used in the dataset is 'US', in order to clean the data, this consistent format has been applied to all relevant fields.
  file reference: audit.py*

3. Auditing Post Codes (attribute: 'addr:postcode')

- To clean the data, the code has been limited to the first 5 digits as a standard
- Irrelevant numbers have been removed
- In some fields, the postal code is prefixed with the state name, in which case 'CA' has been removed
  file reference: audit.py*

4. Auditing Street Names (attribute: 'addr:street')

- Various formats have been used to represent street names (eg. Boulevard vs Blvd.)
- To clean the data, all such cases have been handled by mapping the abbreviated name to the correct format that needs to be followed.
  file reference: audit.py*

5. Auditing State Name (attribute: 'addr:state')

- The state name has several variations such as 'CA', 'Ca', 'California', 'ca', 'california'
- For data cleaning, a standard format is being followed, which is state = CA
  file reference: audit.py*

After auditing, the file was cleaned and changes were written to a new file (san-francisco-modified.osm)
file reference: audit.py*

```
processTags(xmlFileName, writeToFile=True)
```

```
#Tags fixed with this script:
-------------  -----
addr:country   58144
addr:city      57826
addr:postcode  16586
addr:street     9597
addr:state     23220
-------------  -----
Total         165373
Generating XML san-francisco-modified.osm
```

Data Overview using SQL

This section provides a statistical overview of the dataset, MySql queries used to gather them, and a few additional details about the data.

1. File size

Filename	Description	Size
san-francisco.osm	original XML file	1.4GB
san-francisco-modified.osm	cleaned XML file	1.4GB
san-francisco_sample.osm	sample file(from cleaned xml file)	9.4MB
nodes.csv	csv file 1	553.6MB
nodes_tags.csv	csv file 2	9.3MB
ways.csv	csv file 3	50.1MB
ways_tags.csv	csv file 4	54.6MB
ways_nodes.csv	csv file 5	188.3MB

Database Statistics

```
SELECT table_name, table_rows, data_length, index_length, round(((data_length
FROM information_schema.TABLES
WHERE table_schema = "sf";
```

```
+------------+------------+-------------+--------------+------------+
| table_name | table_rows | data_length | index_length | Size in MB |
+------------+------------+-------------+--------------+------------+
| nodes      |    6390417 |   610271232 |            0 |     582.00 |
| nodes_tags |     251699 |    16269312 |      5783552 |      21.03 |
| ways       |     815969 |    62488576 |            0 |      59.59 |
| ways_nodes |    7634315 |   343769088 |    410746880 |     719.56 |
| ways_tags  |    1653952 |   100270080 |     33112064 |     127.20 |
+------------+------------+-------------+--------------+------------+
```

2. Number of unique users

```
SELECT COUNT(DISTINCT t1.uid) AS unique_users
FROM
    (SELECT uid FROM nodes UNION SELECT uid FROM ways) t1;
```

```
+--------------+
| unique_users |
+--------------+
|         2732 |
+--------------+
```

3. Top 10 contributing users

```
SELECT t1.uid, t1.user, COUNT(*) AS times_visited
FROM
    (SELECT uid, user FROM nodes
     UNION ALL
     SELECT uid, user FROM ways) t1
GROUP BY t1.uid, t1.user
ORDER BY times_visited DESC
LIMIT 10;
```

```
+---------+-------------+---------------+
| uid     | user        | times_visited |
+---------+-------------+---------------+
|   94578 | andygol     |       1496763 |
| 1240849 | ediyes      |        887785 |
| 1829683 | Luis36995   |        675919 |
| 2226712 | dannykath   |        545991 |
| 2219338 | RichRico    |        415806 |
|  510836 | Rub21       |        383523 |
| 2511706 | calfarome   |        190825 |
|  169004 | oldtopos    |        165938 |
|   14293 | KindredCoda |        148831 |
| 2748195 | karitotp    |        139874 |
+---------+-------------+---------------+
```

4. Number of users with only 1 contribution

```
SELECT COUNT(*) AS number_of_users
FROM
    (SELECT t1.uid, COUNT(*) AS times_visited
    FROM
        (SELECT uid FROM nodes
         UNION ALL
         SELECT uid FROM ways) t1
    GROUP BY t1.uid
    HAVING times_visited = 1) t2;
```

```
+-----------------+
| number_of_users |
+-----------------+
|             689 |
+-----------------+
```

5. Number of nodes

```
SELECT COUNT(*) AS number_of_nodes
FROM nodes;
```

```
+-----------------+
| number_of_nodes |
+-----------------+
|         6600627 |
+-----------------+
```

6. Number of ways

```
SELECT COUNT(*) AS number_of_ways
FROM ways;
```

```
+----------------+
| number_of_ways |
+----------------+
|         819571 |
+----------------+
```

7. Top 3 node attributes to which the most contributions are made

```
SELECT DISTINCT `key` AS k, COUNT(*) AS count
FROM nodes_tags
GROUP BY k
ORDER BY count DESC
LIMIT 3;
```

```
+-------------+-------+
| k           | count |
+-------------+-------+
| highway     | 27775 |
| housenumber | 24294 |
| street      | 21695 |
+-------------+-------+
```

8. Top 3 way attributes to which the most contributions are made

```
SELECT DISTINCT `key` AS k, COUNT(*) AS count
FROM ways_tags
GROUP BY k
ORDER BY count DESC
LIMIT 3;
```

```
+----------+--------+
| k        | count  |
+----------+--------+
| building | 670983 |
| height   | 143329 |
| highway  | 113810 |
+----------+--------+
```

Additional Ideas

From the above statistics, since we know that the 'highway' attribute is frequently updated, we will explore this attribute in further detail.

1. Number of speed cameras that have been identified and recorded

```
SELECT COUNT(*) AS number_of_speed_cameras
FROM nodes_tags
WHERE `key` = 'highway' AND value = 'speed_camera';
```

```
+-------------------------+
| number_of_speed_cameras |
+-------------------------+
|                       5 |
+-------------------------+
```

2. Number of constructions going on along the highway

```
SELECT COUNT(*) AS number_of_constructions
FROM ways_tags
WHERE `key` = 'highway' AND value = 'construction';
```

```
+-------------------------+
| number_of_constructions |
+-------------------------+
|                      37 |
+-------------------------+
```

We could check if the constructions listed above are up to date. The assumption is that some of the 37 constructions may have been completed and some new ones might not have been updated yet.

- To improve the OSM data further, one strategy could be to set up a validation system for entries like postal code. This would reduce the amount of cleaning required for anyone trying to analyze the data. However, there could be potential problems to doing this:
    - if user input involves the capability to tag a new place, and not knowing the postal code, if the user enters some code which is incorrect but still valid, we would again be looking at unclean data

---