OpenStreetMap Data Case Study

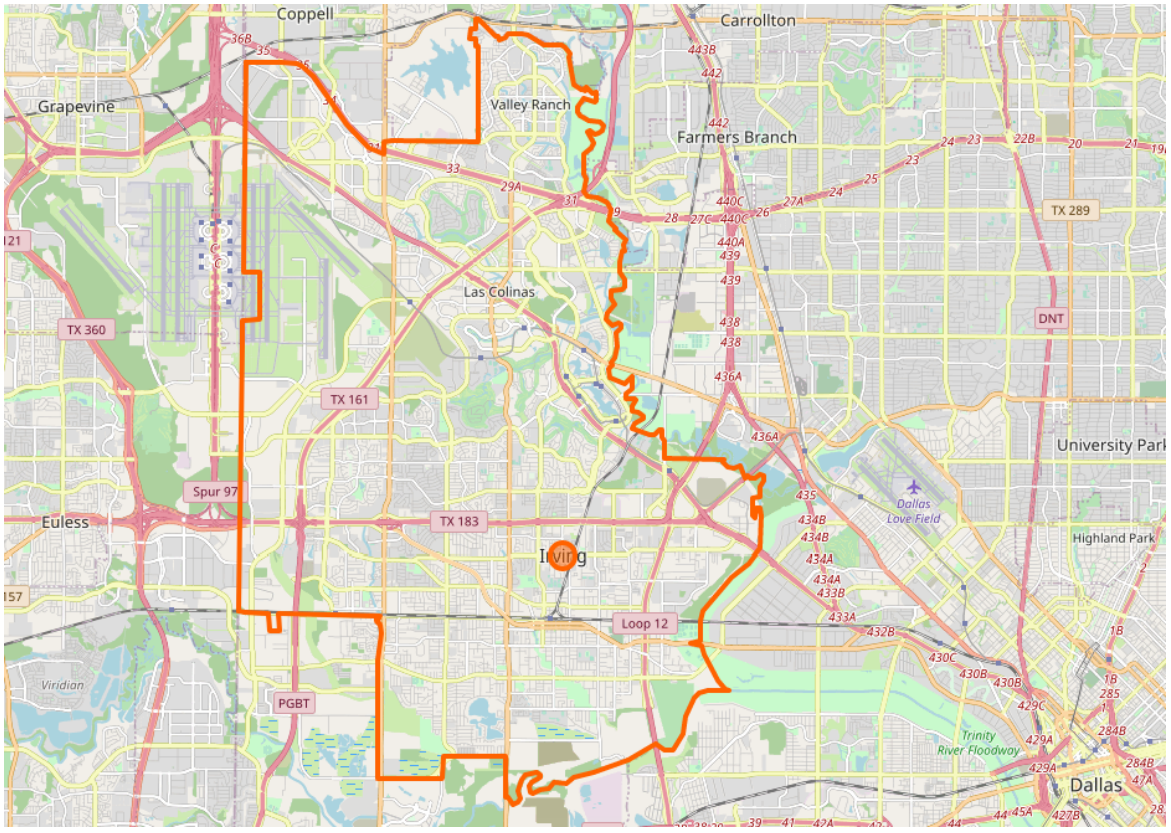
Udacity Project Overview

OpenStreetMap (OSM) is built by a community of mappers that contribute and maintain data about roads, trails, cafés, railway stations, and much more, all over the world. This project is part of Udacity's Data Analyst Nanodegree course for Data Wrangling. The main point of this project is to learn how to access data and perform data cleaning with Python. To start, I downloaded the OSM data first. Then, I accessed the XML data from the downloaded OpenStreet Map file. Next, I audited several fields and performed some data cleaning while parsing the OSM into to five CSV files before importing the files to SQL tables. Last, the cleaned data will then be imported into a DBMS or MongoDB. In this case, I used a DBMS and ran a few SQL queries on the cleaned data to complete the project.

OpenStreet Map Area

Irving, TX, United States

- https://www.openstreetmap.org/relation/5748832



> I selected the above area, Irving, Texas, for this project because this is where I've lived for many years now in the Dallas suburb of Irving, Texas. Therefore, I was interested in this area's OSM dataset by examaning the tags attached to its basics data structures (nodes, ways, and relations) found in the OSM file.

Problems Encountered in the OSM File

After initially downloading a small sample size of the city of Irving, Texas, and running it against a provisional data.py file, I noticed the following key problems with the data.

- Address contained abbreviated street name *(e.g. "N Macarthur Blvd")*
- Address contained special characters *(e.g. brackets '[]', commas ',', hyphens '-', periods '.', etc.)*
- Key or "k" tags contained values in different languages such as Urdu, Hindi, etc.
- Secondlevel `"k"` tags with the value `"type"` (which overwrites the element's previously processed `node["type"]field`).
- Street names in second level `"k"` tags pulled from Tiger GPS data and divided into segments, in the following format:

```
<tag k="tiger:name_base" v="16th;Burleson" />
<tag k="tiger:name_direction_prefix" v="NW" />
<tag k="tiger:name_type" v="St" />
```

Problematic Characters

Certain tags were found to contain problematic characters. These characters can result in troublesome data values that are either hard to read or process programatically. In order to exclude such tags from the dataset, a python regular expression was used.

```
re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')
```

The actual regular expression `[=\+/&<>;\'"\?%#$@\,\. \t\r\n]` looks for new lines, tabs, commas, single/double quotes, semi-colons, equal signs or characters like %,?,%,$,@,#. If any of these characters are found in the tags key, it is not included.

Function for Correcting Street Names

The function **update_name(name, mapping)** is used to remove the problems encountered during the cleaning process. It rectifies the potential for human errors caused in data entries.

```
# This function is used for cleaning the street names
def update_name(name, mapping):

        unwanted = ['(',')','/', '[', ']', '.', ',']  # List of unwanted characters
    cname = ''                  # Create an empty string

    # for loop to remove unwanted characters
    for i in range(len(name)):
        if name[i] not in unwanted:
            cname = cname + name[i]

    # Slicing to remove '-'
    if cname[0]=='-':
        cname = cname[1:]
    if cname[-1]=='-' or cname[-1]==',':
        cname = cname[:-1]

    # To remove postal codes from street name
    if '-' in cname:
        ch = cname.split('-')
        if len(ch[1])>=4:
            cname = ch[0]

    # Capitalize the first letter of each street name and convert other letters to lower case
    low_name = cname.lower()
    if ' ' in low_name:
        cname = ''
        t = low_name.split(' ')
        for i in t:
            cname = cname + ' ' + i.capitalize()
    else:
        cname = low_name.capitalize()

    # Mapping
```

```
    k = mapping.keys()
    key_list = list(k)
    for abrev in key_list:
        if abrev in cname.split():
            cname = cname.replace(abrev,mapping[abrev])

    return cname
```

Note: For the Mapping values, I created the following list of problematic names that were identified and names that may potentially cause issues into a dictionary for cleaning.

```
{"Avenue", "Boulevard", "Bridge", "Center", "Circle", "Court", "Drive", "East", "Expressway", "Freeway", "Highway", "Lane", "Loop", "I
```

> This updated all substrings in problematic address strings, such that: *“w Southlake Blvd”* becomes *“West Southlake Boulevard”*

Sort cities by count, descending

```
sqlite> SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key = 'city'
GROUP BY tags.value
ORDER BY count DESC;
```

The following are the city results that were sorted by count and Tag.value for readability:

```
Tag.value                       Count
Dallas                          139
Irving                          33
Grapevine                       20
Grand Prairie                   14
Carrollton                      12
Southlake                       12
Coppell                         11
Colleyville                     10
Fort Worth                      9
Addison                         8
Euless                          8
Arlington                       7
Richardson                      5
Westlake                        5
Hurst                           4
Farmers Branch                  3
Bedford                         2
DFW Airport                     2
Keller                          2
North Richland Hills            1
University Park                 1
```

> These results confirmed my suspicion that this metro extract would perhaps be more aptly named “Dallas/Fort Worth Metropolitan Area” for its inclusion of surrounding cities in the sprawl.

Converting the Data into CSV Files

After the cleaning was completed, CSV files were created from the cleaned data in the OSM files using python script **csv-to-db.py**. The CSV files are namely nodes.csv, nodes_tags.csv, ways.csv, ways_tags.csv and ways_nodes.csv.

Afterwards, the csv files were imported into a database called **irving_texas.db** as tables with the names as nodes, nodes_tags, ways, ways_tags and ways_nodes. Here is the following DBMS schema.

```
schema = {
    'node': {
```

```
        'type': 'dict',
        'schema': {
            'id': {'required': True, 'type': 'integer', 'coerce': int},
            'lat': {'required': True, 'type': 'float', 'coerce': float},
            'lon': {'required': True, 'type': 'float', 'coerce': float},
            'user': {'required': True, 'type': 'string'},
            'uid': {'required': True, 'type': 'integer', 'coerce': int},
            'version': {'required': True, 'type': 'string'},
            'changeset': {'required': True, 'type': 'integer', 'coerce': int},
            'timestamp': {'required': True, 'type': 'string'}
        }
    },
    'node_tags': {
        'type': 'list',
        'schema': {
            'type': 'dict',
            'schema': {
                'id': {'required': True, 'type': 'integer', 'coerce': int},
                'key': {'required': True, 'type': 'string'},
                'value': {'required': True, 'type': 'string'},
                'type': {'required': True, 'type': 'string'}
            }
        }
    },
    'way': {
        'type': 'dict',
        'schema': {
            'id': {'required': True, 'type': 'integer', 'coerce': int},
            'user': {'required': True, 'type': 'string'},
            'uid': {'required': True, 'type': 'integer', 'coerce': int},
            'version': {'required': True, 'type': 'string'},
            'changeset': {'required': True, 'type': 'integer', 'coerce': int},
            'timestamp': {'required': True, 'type': 'string'}
        }
    },
    'way_nodes': {
        'type': 'list',
        'schema': {
            'type': 'dict',
            'schema': {
                'id': {'required': True, 'type': 'integer', 'coerce': int},
                'node_id': {'required': True, 'type': 'integer', 'coerce': int},
                'position': {'required': True, 'type': 'integer', 'coerce': int}
            }
        }
    },
    'way_tags': {
        'type': 'list',
        'schema': {
            'type': 'dict',
            'schema': {
                'id': {'required': True, 'type': 'integer', 'coerce': int},
                'key': {'required': True, 'type': 'string'},
                'value': {'required': True, 'type': 'string'},
                'type': {'required': True, 'type': 'string'}
            }
        }
    }
}
```

Data Overview and Additional Ideas

Files size

```
irving_texas_osm.zip (compressed) -------------- 62.47 MB
irving_texas.osm ------------------------------- 803.58 MB
irving_texas.db (database)---------------------- 47.24 MB
nodes.csv -------------------------------------- 32.93 MB
nodes_tags.csv --------------------------------- 0.27 MB
ways.csv --------------------------------------- 3.37 MB
ways_tags.csv ---------------------------------- 4.11 MB
ways_nodes.csv --------------------------------- 10.13 MB
sample_irving_texas.osm (in repo) -------------- 85.15 MB
Irving-TX-area.png ----------------------------- 0.86 MB
```

Postal Codes

Postal codes contained 5digit zip codes. There were no issues with the postal codes after grouping together with the following aggregator:

```
SELECT tags.value AS postCode, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
          UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key='postcode'
GROUP BY tags.value
ORDER BY count DESC;
```

The following are the top ten postal code results, beginning with the highest count:

```
postCode        count
76051           19
76053           17
76092           12
75050           12
75006           12
75019           11
76039           10
76034           10
75205           10
75001           9
```

Number of nodes

```
--Number of nodes
SELECT COUNT(*)
  FROM nodes;
```

```
364223
```

Number of ways

```
--Number of ways
SELECT COUNT(*)
  FROM ways;
```

```
50537
```

Number of unique users

```
--Number of ways
SELECT COUNT(*) FROM ways;
SELECT COUNT(DISTINCT (e.uid) ) AS unique_users
  FROM (
           SELECT uid
             FROM nodes
           UNION ALL
           SELECT uid
             FROM ways
       )
       AS e;
```

```
1408
```

Top 10 contributing users

```
--Top 10 contributing users
SELECT e.user,
       COUNT( * ) AS num
  FROM (
           SELECT user
             FROM nodes
           UNION ALL
           SELECT user
             FROM ways
```

```
        )
        AS e
 GROUP BY e.user
 ORDER BY num DESC
 LIMIT 10;
```

user	num
Andrew Matheny_import	234246
Andrew Matheny	55462
TheDude05_import	11362
woodpeck_fixbot	10590
Stephen214	7299
Mark@MJS	5697
j5f8k	3540
fmmute	3080
MapRogers	2523
Jame Retief	2217

Number of users appearing only once (having 1 post)

```
--Number of users appearing only once (having 1 post)
SELECT COUNT( * ) AS users_with_OnePost
  FROM (
           SELECT e.user,
                  COUNT( * ) AS num
             FROM (
                      SELECT user
                        FROM nodes
                      UNION ALL
                      SELECT user
                        FROM ways
                  )
                  e
            GROUP BY e.user
           HAVING num = 1
       )
      AS u;
```

```
219
```

Additional Ideas

Contributor Statistics and Suggestions

The contributions of users seems incredibly skewed, possibly due to automated versus manual map editing. The following are some user percentage statistics:

- Top user contribution percentage (“Andrew Matheny_import”) 56.57%
- Combined top 2 users' contribution (“Andrew Matheny_import” and “Andrew Matheny”) 69.97%
- Combined Top 10 users contribution 80.61%
- Combined number of users making up only 1% of posts 219

In the context of the OpenStreetMap, I agree that if user data were more prominently displayed, perhaps others would take an initiative in submitting more edits to the map. In addition, if everyone sees that only a handful of power users are creating more than a majority of a given map. Maybe offering incentives,rewards, badges, or competing for a position in a leaderboard will help.

Another suggestion that I think it would be great is if each tag in the OSM file contained an associated confidence score. The motivation behind this is to assure that other correctors or analyst can ignore particular data and focus on auditing or cleaning up other data with data issues and low confidence scores. Another benefit is more users can view OSM data as a highly reliable source. However, with this approach, implementing a confidence score become a major and ongoing project. In addition, a cost/benefit analysis would need to be conducted to weigh the benefits and negative impacts to implementing this model.

Additional Data Exploration

Top 10 Amenities

```
--Top 10 appearing amenities
SELECT value,
       COUNT( * ) AS num
  FROM nodes_tags
 WHERE [key] = 'amenity'
 GROUP BY value
 ORDER BY num DESC
 LIMIT 10;
```

value	num
parking_entrance	70
restaurant	42
place_of_worship	35
bench	30
fast_food	24
fountain	17
cafe	17
waste_disposal	13
waste_basket	8
bar	5

Top Religion

```
--Biggest religion (no surprise here)
SELECT nodes_tags.value,
       COUNT( * ) AS num
  FROM nodes_tags
       JOIN
       (
           SELECT DISTINCT (id)
             FROM nodes_tags
            WHERE value = 'place_of_worship'
       )
       i ON nodes_tags.id = i.id
 WHERE nodes_tags.[key] = 'religion'
 GROUP BY nodes_tags.value
 ORDER BY num DESC
 LIMIT 1;
```

christian 34

Most popular cuisines

```
--Most popular cuisines
SELECT nodes_tags.value,
       COUNT( * ) AS num
  FROM nodes_tags
       JOIN
       (
           SELECT DISTINCT (id)
             FROM nodes_tags
            WHERE value = 'restaurant'
       )
       i ON nodes_tags.id = i.id
 WHERE nodes_tags.[key] = 'cuisine'
 GROUP BY nodes_tags.value
 ORDER BY num DESC;
```

value	num
pizza	4
asian	4
american	4
sushi	2
regional	2
mexican	2
thai	1
tex-mex;mexican	1
tex-mex	1
sandwich	1
italian,pizza	1
italian	1
chinese	1
cajun	1
burger	1

Conclusion

After cleaning the OSM data of Irving, Texas, it was clear that the area I chosen for this project is incomplete. However, it's exciting to see contributors continue to grow year over year. Also, I believe it has been well cleaned for the purposes of this exercise. During the cleaning process, I programmatically cleaned up inconsistent street types and abbreviated street names. Next, I iteratively wrote the cleaned OSM data to CSV files before loading the files into a SQL database. Finally, I wrote database queries to review statistical information as well as to understand the structure of the data set.

As mentioned previously, a suggestion for improving the data and its analysis is to ensure accurate data is being entered by contributors. I would recommend adding a confidence level or score to each data that was entered. In addition, for data consistency and integrity, I believe OSM should implement data quality rules that restrict what data can be entered. Last, I would also recommend creating a leaderboard or an incentive program to encourage more users to contribute to OpenStreetMap.org. Overall, this was a very challenging and fun assignment.