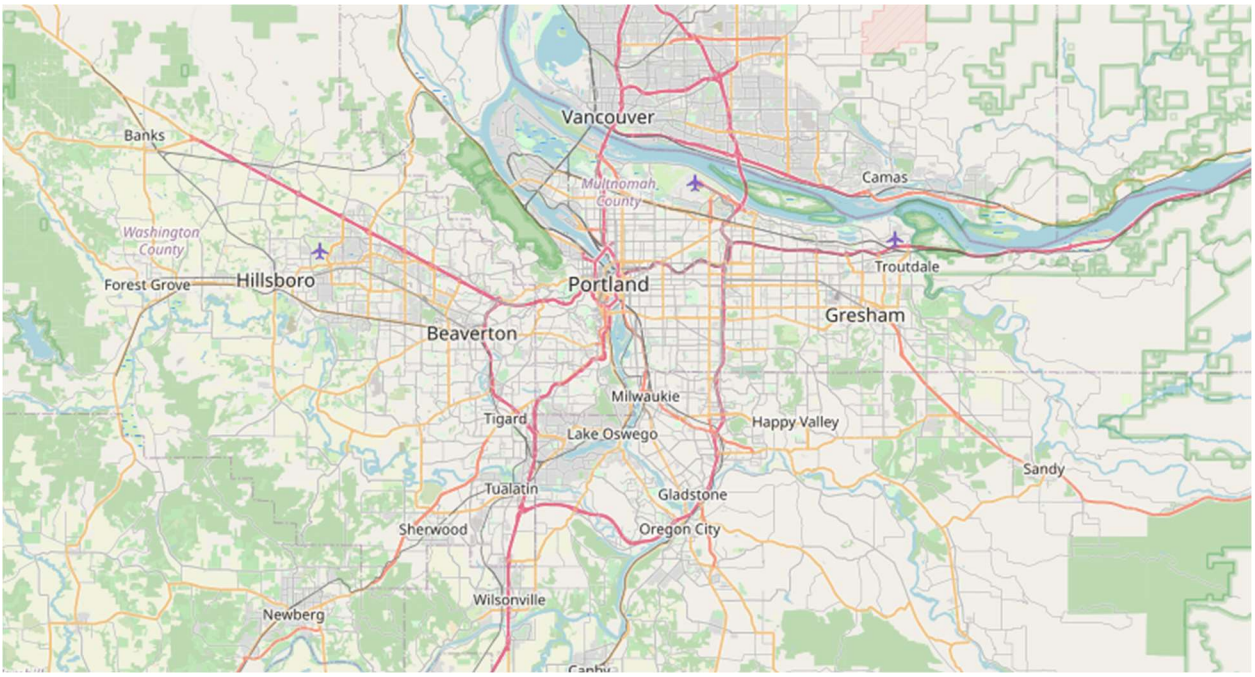
# Wrangling Data with SQL

Student/Analyst: Christopher Ackerman

[Relation: Portland (186579) | OpenStreetMap]




## Analysis Introduction:

Under this project, OpenStreetMap was used to retrieve xml data via their APIs that describes the map of Portland OR. In this analysis, various data wrangling methods and tools were used to wrangle, parse, and clean the data provided from OpenStreetMap.

Microsoft SQL Server was the method of choice for reviewing this data.

As mentioned, Portland Oregon was my choice of area from OpenStreetMap. The reason I choose this City, is because it is one of the most walkable Cities in the U.S. and I find this as fun fact.

# 1. Data Quality Auditing

## 1.1 Inconsistent Street Types:

Examples: “Ave”, “ave”, “Ave”, ave.”

The street types of addresses in the dataset were inconsistent with their abbreviations, and lower/upper cases. By auditing the street types, a function to map different types of street type abbreviations and lower/upper cases to non-abbreviated street types with first letter capitalized (e.g. "Street", "Avenue") was made to resolve this issue.

**audit.py**

```
11
12  # Auditing and cleaning street names
13
14  OSMFILE = "portland_oregonosm.xml"
15  street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
16
17  street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
18
19  expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
20              "Trail", "Parkway", "Commons", "Cove", "Alley", "Park", "Way", "Walk" "Circle", "Highway",
21              "Plaza", "Path", "Center", "Mission"]
22
23  mapping = { "Ave": "Avenue",
24              "Ave.": "Avenue",
25              "avenue": "Avenue",
26              "ave": "Avenue",
27              "Blvd": "Boulevard",
28              "Blvd.": "Boulevard",
29              "Blvd,": "Boulevard",
30              "Boulavard": "Boulevard",
31              "Boulvard": "Boulevard",
32              "Ct": "Court",
33              "Dr": "Drive",
34              "Dr.": "Drive",
35              "E": "East",
36              "Hwy": "Highway",
37              "Ln": "Lane",
38              "Ln.": "Lane",
39              "Pl": "Place",
40              "Plz": "Plaza",
41              "Rd": "Road",
42              "Rd.": "Road",
43              "St": "Street",
44              "St.": "Street",
45              "st": "Street",
46              "street": "Street",
47              "square": "Square",
48              "parkway": "Parkway"
49              }
50
51
52  def audit_street_type(street_types, street_name):
53      m = street_type_re.search(street_name)
54      if m:
55          street_type = m.group()
56          if street_type not in expected:
57              street_types[street_type].add(street_name)
58
59
60  def street_name(elem):
61      return (elem.attrib['k'] == "addr:street")
62
63
64  def audit(osmfile):
65      osm_file = open(osmfile, "r")
66      street_types = collections.defaultdict(set)
67      for event, elem in ET.iterparse(osm_file, events=("start",)):
68
69          if elem.tag == "node" or elem.tag == "way":
70              for tag in elem.iter("tag"):
71                  if street_name(tag):
72                      audit_street_type(street_types, tag.attrib['v'])
73      osm_file.close()
74      return street_types
75
76
77  def update_name(name, mapping, regex):
78      m = regex.search(name)
79      if m:
80          st_type = m.group()
81          if st_type in mapping:
82              name = re.sub(regex, mapping[st_type], name)
83      return name
84
```

## 2. Overview of the Data

### 2.1 Parsing of Data – Prepping for SQL Server

The data.py script provided was modified for parsing the elements in the XML file after the cleansing of the data's street names.
I transformed the elements from xml document format to tabular format, eventually into csv files. Paving the way to import the csv files into SQL Server as tables for analysis.

- Refer to last pages for final code[1].

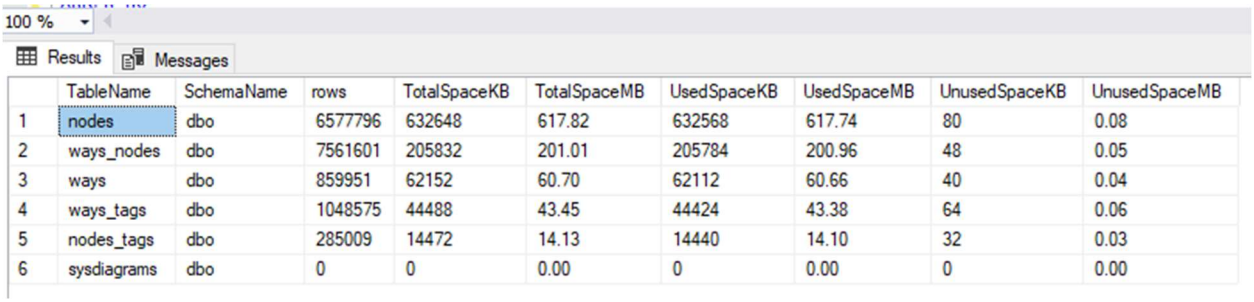
## 3. Data Overview

### 3.1 File Sizes:

.csv files

Nodes_tags: 10Mb
Nodes: 603Mb
Ways: 176Mb
Ways_tags: 152Mb

DB Table sizes


100 %

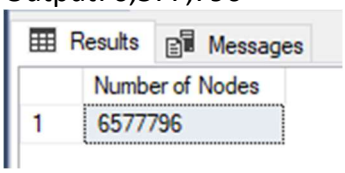
Results | Messages

| | TableName | SchemaName | rows | TotalSpaceKB | TotalSpaceMB | UsedSpaceKB | UsedSpaceMB | UnusedSpaceKB | UnusedSpaceMB |
|---|---|---|---|---|---|---|---|---|---|
| 1 | nodes | dbo | 6577796 | 632648 | 617.82 | 632568 | 617.74 | 80 | 0.08 |
| 2 | ways_nodes | dbo | 7561601 | 205832 | 201.01 | 205784 | 200.96 | 48 | 0.05 |
| 3 | ways | dbo | 859951 | 62152 | 60.70 | 62112 | 60.66 | 40 | 0.04 |
| 4 | ways_tags | dbo | 1048575 | 44488 | 43.45 | 44424 | 43.38 | 64 | 0.06 |
| 5 | nodes_tags | dbo | 285009 | 14472 | 14.13 | 14440 | 14.10 | 32 | 0.03 |
| 6 | sysdiagrams | dbo | 0 | 0 | 0.00 | 0 | 0.00 | 0 | 0.00 |

### 3.2 Number of nodes:

```
SELECT COUNT(*) AS 'Number of Nodes'
FROM nodes
```

Output: 6,577,796

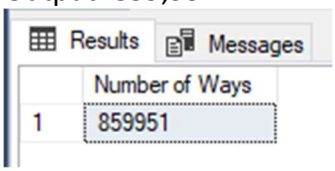

Results | Messages

| | Number of Nodes |
|---|---|
| 1 | 6577796 |

3.3 Number of ways:

```
SELECT COUNT(*) AS 'Number of Ways'
FROM ways
```

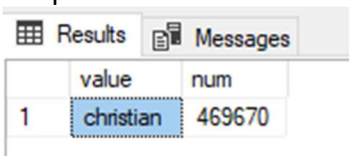
Output: 859,951




3.4 Most Popular Religion:

```
SELECT TOP 1 nodes_tags.value, COUNT(*) as num
FROM nodes_tags
INNER JOIN
(
SELECT DISTINCT id
FROM nodes_tags
WHERE value='place_of_worship'
) i
ON nodes_tags.[key] = 'religion'
GROUP BY nodes_tags.value
ORDER BY num DESC
```

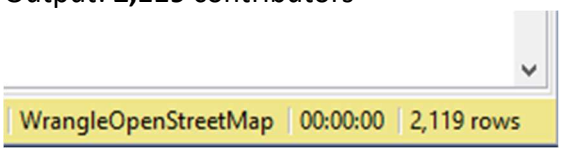
Output




3.5 Number of unique users:

```
SELECT DISTINCT uid
FROM nodes
UNION ALL
SELECT DISTINCT uid
FROM ways
```

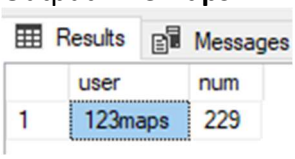
Output: **2,119** contributors



3.6 Top Contributing user:

```
SELECT TOP 1 u.[user], COUNT(*) as num
FROM
(
SELECT [user]
FROM nodes
UNION ALL
SELECT [user]
FROM ways) u
GROUP BY u.[user]
```

Output: **123maps**

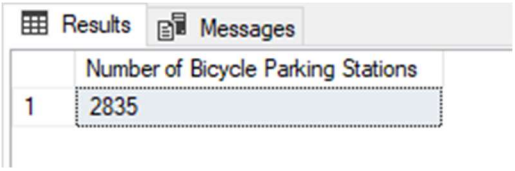



3.6 Number of Train Stations:

Since Portland OR is rated as one of the most sustainable Cities in The States in 2015. I wanted to find out how many bicycle parking stations there are.

```
SELECT COUNT(*) 'Number of Bicycle Parking Stations'
FROM nodes_tags
WHERE value='bicycle_parking'
```

Output: 2,835 << That's a lot of places to park a bike!



# 4. Conclusion and Feedback

## 4.1 Rating System:

**User Ratings**
One piece of crucial information missing from the dataset is the ratings of places. By incorporating a node tag with user ratings can help user answer questions such as "What are some of the best restaurants in town?"

One way to gather this rating information:

1. User contribution. It is easy to implement this, the problem is the number of active contributing users for our OpenStreetMap data is low, the ratings will not have a sample size large enough to be representative.

## 4.2 Conclusion:

This analysis of OpenStreetMap Portland OR has helped me dig into the problems and inconsistency of the OpenStreetMap data. I found it cool to review the different kind of statistics that related to it's sustainability.

After cleaning the dataset, I imported this dataset into a SQL database for further exploration. I obtained some statistics and answered some questions using TSQL queries, I really liked this project.

# Reference to Section 2.1[1].

**Data.py**

```
import csv
import codecs
import re
import xml.etree.cElementTree as ET
from unittest import TestCase

import cerberus
import schema

OSM_PATH = "portland_oregonosm.xml"

NODES_PATH = "nodes.csv"
NODE_TAGS_PATH = "nodes_tags.csv"
WAYS_PATH = "ways.csv"
WAY_NODES_PATH = "ways_nodes.csv"
WAY_TAGS_PATH = "ways_tags.csv"

LOWER_COLON = re.compile(r'^([a-z]|_)+:([a-z]|_)+')
PROBLEMCHARS = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

SCHEMA = schema.schema

NODE_FIELDS = ['id', 'lat', 'lon', 'user', 'uid', 'version', 'changeset', 'timestamp']
NODE_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_FIELDS = ['id', 'user', 'uid', 'version', 'changeset', 'timestamp']
WAY_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_NODES_FIELDS = ['id', 'node_id', 'position']


def shape_element(element, node_attr_fields=NODE_FIELDS, way_attr_fields=WAY_FIELDS,
                  problem_chars=PROBLEMCHARS, default_tag_type='regular'):
    """Clean and shape node or way XML element to Python dict"""

    node_attribs = {}
    way_attribs = {}
    way_nodes = []
    tags = []

    if element.tag == 'node':
        for attrib in element.attrib:
            if attrib in NODE_FIELDS:
                node_attribs[attrib] = element.attrib[attrib]

        for child in element:
            node_tag = {}
            if LOWER_COLON.match(child.attrib['k']):
                node_tag['type'] = child.attrib['k'].split(':',1)[0]
                node_tag['key'] = child.attrib['k'].split(':',1)[1]
                node_tag['id'] = element.attrib['id']
                node_tag['value'] = child.attrib['v']
                tags.append(node_tag)
            elif PROBLEMCHARS.match(child.attrib['k']):
                continue
            else:
```

```
            node_tag['type'] = 'regular'
            node_tag['key'] = child.attrib['k']
            node_tag['id'] = element.attrib['id']
            node_tag['value'] = child.attrib['v']
            tags.append(node_tag)

        return {'node': node_attribs, 'node_tags': tags}

    elif element.tag == 'way':
        for attrib in element.attrib:
            if attrib in WAY_FIELDS:
                way_attribs[attrib] = element.attrib[attrib]

        position = 0
        for child in element:
            way_tag = {}
            way_node = {}

            if child.tag == 'tag':
                if LOWER_COLON.match(child.attrib['k']):
                    way_tag['type'] = child.attrib['k'].split(':',1)[0]
                    way_tag['key'] = child.attrib['k'].split(':',1)[1]
                    way_tag['id'] = element.attrib['id']
                    way_tag['value'] = child.attrib['v']
                    tags.append(way_tag)
                elif PROBLEMCHARS.match(child.attrib['k']):
                    continue
                else:
                    way_tag['type'] = 'regular'
                    way_tag['key'] = child.attrib['k']
                    way_tag['id'] = element.attrib['id']
                    way_tag['value'] = child.attrib['v']
                    tags.append(way_tag)

            elif child.tag == 'nd':
                way_node['id'] = element.attrib['id']
                way_node['node_id'] = child.attrib['ref']
                way_node['position'] = position
                position += 1
                way_nodes.append(way_node)

        return {'way': way_attribs, 'way_nodes': way_nodes, 'way_tags': tags}


# ================================================== #
#               Helper Functions                     #
# ================================================== #
def get_element(osm_file, tags=('node', 'way', 'relation')):
    """Yield element if it is the right type of tag"""

    context = ET.iterparse(osm_file, events=('start', 'end'))
    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()


def validate_element(element, validator, schema=SCHEMA):
```

```
    """Raise ValidationError if element does not match schema"""
    if validator.validate(element, schema) is not True:
        field, errors = next(validator.errors.items())
        message_string = "\nElement of type '{0}' has the following errors:\n{1}"
        error_strings = (
            "{0}: {1}".format(k, v if isinstance(v, str) else ", ".join(v))
            for k, v in errors.items()
        )
        raise cerberus.ValidationError(
            message_string.format(field, "\n".join(error_strings))
        )


class UnicodeDictWriter(csv.DictWriter, object):
    """Extend csv.DictWriter to handle Unicode input"""

    def writerow(self, row):
        super(UnicodeDictWriter, self).writerow({
            k: v for k, v in row.items()
        })

    def writerows(self, rows):
        for row in rows:
            self.writerow(row)


# ================================================== #
#               Main Function                        #
# ================================================== #
def process_map(file_in, validate):
    """Iteratively process each XML element and write to csv(s)"""

    with codecs.open(NODES_PATH, 'w', 'utf-8') as nodes_file, \
         codecs.open(NODE_TAGS_PATH, 'w', 'utf-8') as nodes_tags_file, \
         codecs.open(WAYS_PATH, 'w', 'utf-8') as ways_file, \
         codecs.open(WAY_NODES_PATH, 'w', 'utf-8') as way_nodes_file, \
         codecs.open(WAY_TAGS_PATH, 'w', 'utf-8') as way_tags_file:

        nodes_writer = UnicodeDictWriter(nodes_file, NODE_FIELDS)
        node_tags_writer = UnicodeDictWriter(nodes_tags_file, NODE_TAGS_FIELDS)
        ways_writer = UnicodeDictWriter(ways_file, WAY_FIELDS)
        way_nodes_writer = UnicodeDictWriter(way_nodes_file, WAY_NODES_FIELDS)
        way_tags_writer = UnicodeDictWriter(way_tags_file, WAY_TAGS_FIELDS)

        nodes_writer.writeheader()
        node_tags_writer.writeheader()
        ways_writer.writeheader()
        way_nodes_writer.writeheader()
        way_tags_writer.writeheader()

        validator = cerberus.Validator()

        for element in get_element(file_in, tags=('node', 'way')):
            el = shape_element(element)
            if el:
                if validate is True:
                    validate_element(el, validator)

                if element.tag == 'node':
```

```
                nodes_writer.writerow(el['node'])
                node_tags_writer.writerows(el['node_tags'])
            elif element.tag == 'way':
                ways_writer.writerow(el['way'])
                way_nodes_writer.writerows(el['way_nodes'])
                way_tags_writer.writerows(el['way_tags'])
print("DONE!!")

if __name__ == '__main__':
    process_map(OSM_PATH, validate=True)
```

## Resources and References for Wrangle OpenStreetMap Project:


https://github.com/jeswingeorge/Wrangle-Openstreetmap-data
https://github.com/anilsai/Wrangle-OpenStreetMap-Data-SQL-database-Udacity-project
https://github.com/gauravansal/Wrangle-OpenStreetMap-Data/blob/master/Wrangle%20OpenStreetMap%20Data.ipynb
https://github.com/Zhenmao/udacity-dand-p3-wrangle-openstreetmap-data/blob/master/udacity-dand-p3-wrangle-openstreetmap-data.ipynb
https://github.com/jasonicarter/DAND_OpenStreetMap_Data_MongoDB/tree/master/src
https://github.com/wblakecannon/udacity-dand/tree/master/4-Data-Wrangling/L12-Case-Study-OpenStreetMap-Data
#Used for help on data.py function

https://www.w3schools.com/sql/default.asp
http://wiki.openstreetmap.org/wiki/Map_Features
http://wiki.openstreetmap.org/wiki/OSM_XML
#Links