OpenStreetMap Data Wrangling Project

In this notebook I will summarize my steps and findings towards the Data Wrangling Project course, using data obtained from OpenStreet Map.

Introduction

For this project I have chosen to analyze the OpenStreetMap data for Seattle Washington. I chose this area as I currently am living and working in Seattle.

OpenStreetMap Link - https://www.openstreetmap.org/export#map=14/47.6134/-122.3341 (https://www.openstreetmap.org/export#map=14/47.6134/-122.3341)

Objectives

- Assess the quality of the data for validity, accuracy, completeness, consistency and uniformity.
- Parse and gather data
- Process data
- Learn how to store, query, and aggregate data using SQL.

Data Auditing and cleaning

- I used identifyTags.py to identify the tags used in the datafile.Node and way are the tags I will be looking at.
- TagAudit.py was used to look for tags with only lowercase letters, lowercase letters separated by a colon as well as any problem characters.
- AuditingK.py was used to find the different attributes represented by the 'k' value and measures their occurance.
- I used StreetTypes.py to audit the street names in the seattle.osm file.
- UpdateStreetTypes.py was used to correct inconsistancies with street names in the seattle.osm file.

Tag Identification

This function shows what kind of elements are present in OSM file, and which are important. Reference IdentifyTages.py.

In [2]:
```
import xml.etree.cElementTree as ET
import pprint

OSMFILE = 'Seattle.osm'
SAMPLE_FILE = 'sample.osm'

def count_tags(filename):
    tags = {} #create empty dic to hold values of tags and their counts
    for event, elem in ET.iterparse(filename):
        if elem.tag not in tags.keys():
            tags[elem.tag] = 1
        else:
            tags[elem.tag] += 1
    return tags

tags = count_tags(OSMFILE)
pprint.pprint(tags)
```

```
{'bounds': 1,
 'member': 70807,
 'meta': 1,
 'nd': 251384,
 'node': 217673,
 'note': 1,
 'osm': 1,
 'relation': 1039,
 'tag': 266030,
 'way': 31137}
```

Size of original file

In [3]:
```
# Ref https://stackabuse.com/python-list-files-in-a-directory/

import os
bytes = os.path.getsize('Seattle.osm')
mb = float(bytes / 1000000)
print ("osm file size:", mb, "Mb")
```

```
osm file size: 60.230715 Mb
```

Sample file creation

The following function creats a sample file. Reference sample.py.

In [4]:
```
k = 25 # Parameter: take every k-th top level element. The value was tuned to
 get the appropriate sample file size.

def get_element(filename, tags=('node', 'way', 'relation')):
    context = iter(ET.iterparse(filename, events=('start', 'end')))
    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()

with open(SAMPLE_FILE, 'w',encoding='utf-8') as output:
    output.write('<?xml version="1.0" encoding="UTF-8"?>\n')
    output.write('<osm>\n  ')
    # Write every kth top level element
    for i, element in enumerate(get_element(OSMFILE)):
        if i % k == 0:
            output.write(ET.tostring(element, encoding='utf-8').decode())
    output.write('</osm>')
```

In [5]:
```
# Ref https://stackabuse.com/python-list-files-in-a-directory/

import os
bytes = os.path.getsize('sample.osm')
mb = float(bytes / 1000000)
print ("osm file size:", mb, "Mb")
```

```
osm file size: 2.447847 Mb
```

Auditing the "k" values

I used a function to find the different attributes that are represented by the 'k' value and measures their occurance. Reference AuditingK.py.

I used code to look for tags with only lowercase letters, then lowercase letters separated by a colon and lastly, any problem characters. Reference TagAudit.py.

Street Types Audit

The code found in StreetTypes.py audits the street names in the seattle.osm file.

While I did see a few very small inconsistancies while auditing the street names, I found the formatting to be mostly clean and consistent. The code below attempts to update the street names. Reference UpdateStreetTypes.py.

Prepare Data for SQL

The data is now ready to be imported into SQL. The XML data will be parsed through and converted into tabular format into CSV files. The CSV files can then be imported into sqlite. The code used for this process can be found in sqlprep.py.

Audited changes are carried out when converting to CSV in the script below.

Creating SQL Database

I then use the code found in sqlCreate.py to create the sqlite database.

Exploring the data further with SQL

Number of unique users

In [23]:
```
query='''select count(DISTINCT uid) from nodes; '''

result=cur.execute(query)
for row in result:
    print (row)
```

```
(714,)
```

Number of nodes

In [25]:
```
query='''select count(DISTINCT id) from nodes; '''

result=cur.execute(query)
for row in result:
    print (row)
```

```
(217673,)
```

Number of ways

In [12]:
```
query='''select count(DISTINCT id) from ways; '''

result=cur.execute(query)
for row in result:
    print (row)
```

```
(31137,)
```

Top 5 amenitities

For fun, I took a look at the top 5 amenities in my osm file.

In [22]:
```
query = ("SELECT tags.value, COUNT(*) as count \
            FROM (SELECT * FROM nodes_tags \
            UNION ALL \
            SELECT * FROM ways_tags) tags \
            WHERE tags.key='amenity' \
            GROUP BY tags.value \
            ORDER BY count DESC \
            LIMIT 5")
cur.execute(query)
top_5_amenities = cur.fetchall()
print ("Top 5 amenities:\n")
pprint.pprint(top_5_amenities)
```

```
Top 5 amenities:

[('bicycle_parking', 1400),
 ('parking', 663),
 ('restaurant', 619),
 ('waste_basket', 417),
 ('cafe', 362)]
```

This being Seattle, it is not surprising to see cafes in the top 5 amenities.

In [17]:
```
query = ("SELECT tags.value, COUNT(*) as count \
             FROM (SELECT * FROM nodes_tags \
             UNION ALL \
             SELECT * FROM ways_tags) tags \
             JOIN (SELECT DISTINCT(id) FROM (SELECT * FROM nodes_tags \
             UNION ALL \
             SELECT * FROM ways_tags) WHERE value = 'cafe') as subq \
             ON tags.id = subq.id \
             WHERE tags.key = 'street' \
             GROUP BY tags.value \
             ORDER BY count DESC \
             LIMIT 5")
cur.execute(query)
top_5_caf_street = cur.fetchall()
print ("Top 5 streets by cafe:\n")
pprint.pprint(top_5_caf_street)
```

```
Top 5 streets by cafe:

[('3rd Avenue', 14),
 ('1st Avenue', 13),
 ('East Pike Street', 11),
 ('4th Avenue', 11),
 ('2nd Avenue', 8)]
```

It looks like 3rd Avenue in downtown Seattle is the street with the highest number of cafes.

Conclusion

The datafile I used from OpenStreetMap was surprisingly consistant given the size of the file and the number of users. In general, the file was cleaner than expected.

This project was a good way to learn data gathering, auditing, cleaning and analysis. It was also valuable to learn how to create and import data into an SQL database.

I might suggest that OpenStreetMap somehow provide data validation checks to avoid erroneous data entry into the OSM database.

There are also multiple names for the same types of places. These could be consolidated into one which would save space and give users fewer options to look through. For example, theatre and cinema as well as bar and pub could be combined. Reference improvement.py.

Anticipated problems with suggestions: As usual, data entry by users would be a potential problem which could be resolved through data validation checks.