Wrangle OpenStreetMap Data Project

Project overview

XML OSM dataset of Paris have been downloaded from https://mapzen.com/data/metro-extracts/ (https://mapzen.com/data/metro-extracts/). I chose to analyze Paris, as I recently went there for my holidays and wanted to know if the data conerning this city was consistent. The OSM downloaded is 975Mo. Therefore, it has been parsed with a parameter of k = 100, so that codes could be applied faster. Then, munging techniques have been applied to the data, such as assessing the quality of the data for validity, accuracy, completeness, consistency and uniformity, to clean the OpenStreetMap data. Errors in the dataset have been identified and corrected about:

- Amenities
- Street names
- Postal codes

Then, the file has been converted to CSV and finally, SQL has been used to the data schema to complete the project.

The Dataset

OpenStreetMap's data are structured in well-formed XML documents (.osm files) that consist of the following elements:

- **Nodes**: individual dots used to mark specific locations (such as a postal box). Two or more nodes are used to draw line segments or "ways".
- **Ways**: a line of nodes, displayed as connected line segments. "Ways" are used to create roads, paths, rivers, etc.
- **Relations**: When "ways" or areas are linked in some way but do not represent the same physical thing, a "relation" is used to describe the larger entity they are part of. "Relations" are used to create map features, such as cycling routes, turn restrictions, and areas that are not contiguous. The multiple segments of a long way, such as an interstate or a state highway are grouped into a "relation" for that highway. Another example is a national park with several locations that are separated from each other. Those are also grouped into a "relation".
- **Tags**: description of names, types, ... of elements. Like name or type of roads and other attributes.

Numbers of different elements in the dataset

In [171]:
```
def count_tags(SAMPLE_FILE):
    tags = {}
    for event, elem in ET.iterparse(SAMPLE_FILE):
        if elem.tag in tags:
            tags[elem.tag] += 1
        else:
            tags[elem.tag] = 1
    return tags

tag_view = count_tags(SAMPLE_FILE)
#pprint.pprint(overview_tags)

print("In the dataset, there are the following elements:")
print(" ")
for k, v in tag_view.items():
    print k,v
```

```
In the dataset, there are the following elements:

node 39321
nd 53162
member 1712
tag 31312
relation 217
way 5515
osm 1
```

List of users

In [172]:
```
def process_map(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if 'user' in element.attrib:
            users.add(element.attrib['user'])

    return users

paris_users = process_map(SAMPLE_FILE)
print "There are %s users that added information on Openstreetmap a

print(" ")

for idx, line in enumerate(paris_users):
    if idx >= 10:
        break
    print line
```

```
There are 776 users that added information on Openstreetmap about Pa
ris. These are a sample of 10 users:

OpenBeerMapContributor
Abertzalee
VKaeru
leonremi
Super-Map
JeaRRo
Eric POMMEREAU
Khomille
sevenup
Leonmvd
```

1. List of amenities

In [173]:
```
villages=defaultdict(set)
postcodes=defaultdict(set)
amenities=defaultdict(set)
street=defaultdict(set)
country=defaultdict(set)

for _, elem in ET.iterparse(SAMPLE_FILE, events=("start",)):
    if elem.tag == "node" or elem.tag == "way":
        for tag in elem.iter("tag"):
            if (tag.attrib['k'] == "addr:postcode"):
                postcodes[tag.attrib['v']].add(tag.attrib['v'])
            if (tag.attrib['k'] == "addr:city"):
                villages[tag.attrib['v']].add(tag.attrib['v'])
            if (tag.attrib['k'] == "addr:street"):
                street[tag.attrib['v']].add(tag.attrib['v'])
            if (tag.attrib['k'] == "addr:country"):
                country[tag.attrib['v']].add(tag.attrib['v'])
            if (tag.attrib['k'] == "amenity") :
                amenities[tag.attrib['v']].add(tag.attrib['v'])

print "There are %s amenities in Paris. These are a sample of 10 ar

print(" ")
for idx, line in enumerate(amenities):
    if idx >= 10:
        break
    print line
```

```
There are 55 amenities in Paris. These are a sample of 10 amenities:

taxi
fast_food
marketplace
conservatoire municipal
car_sharing
public_building
cinema
recycling
social_facility
telephone
```

Corrections to the list of amenities

This is the list of the corrected amenities:

```
fast_food => fast_food
marketplace => marketplace
conservatoire municipal => conservatory
car_sharing => car_sharing
public_building => public_building
cinema => cinema
recycling => recycling
telephone => telephone
college => college
parking => parking
```

2. List of street names

This is the list of the different street names in Paris:

```
{'1008949616': 'Rue Jasmin',
 '1789288302': 'Rue des Artistes',
 '2264901172': 'Rue Desnouettes',
 '2996097015': 'Rue Maurice Ripoche',
 '3214241918': 'Rue des Essertes',
 '3214347988': 'Avenue des Tilleuls',
 '386235626': 'Rue Anatole France',
 '714161423': 'Rue de Saintonge',
 '804245689': 'Rue de Vaugirard',
 '840887271': 'Rue de Ponthieu'}
```

List of street types and their frequency

In [179]:
```
from operator import itemgetter

def sort_street_types(street_types):
    result = []
    for key, value in street_types.iteritems():
        result.append((key, len(value)))
        result = sorted(list(result), key=itemgetter(0), reverse=T
    return result

print("These are the 20 most frequent street types and their freque
print(" ")
street_types = sort_street_types(streets)
#print a samle of the list
street_types[:20]
```

```
These are the 20 most frequent street types and their frequency:
```

Out[179]:
```
[('rue', 3),
 ('avenue', 1),
 ('Voie', 1),
 ('Villa', 25),
 ('Square', 14),
 ('Rue', 786),
 ('Route', 4),
 (u'Rond-', 2),
 ('Quai', 15),
 ('Promenade', 1),
 ('Port', 1),
 ('Pont', 3),
 (u'Place', 25),
 ('Passage', 21),
 (u'Parvis', 1),
 ('Le', 1),
 (u'Impasse', 12),
 ('Grande', 1),
 ('Galerie', 1),
 ('Forum', 1)]
```

List of street types and their abbreviations / typo

This section provides a list of street types and similar words in the document. Therefore, typo can be identified.

In [180]:
```
from difflib import get_close_matches

def populate_expected(street_types, threshold):
    expected = []
    for i in street_types[:threshold]:
        expected.append(i[0])

    return expected

EXPECTED = populate_expected(street_types, 20)
EXPECTED


def find_abbreviations(expected, data):
    for i in expected:
        print i, get_close_matches(i, data, 4, 0.5)

find_abbreviations(EXPECTED, list(streets.keys()))

```

```
rue ['rue', 'Rue', 'Route', 'Forum']
avenue ['avenue', 'Avenue', 'Grande']
Voie ['Voie']
Villa ['Villa']
Square ['Square']
Rue ['Rue', 'Route', 'rue']
Route ['Route', 'Rue', 'Autoroute', 'rue']
Rond- [u'Rond-']
Quai ['Quai']
Promenade ['Promenade', 'Grande']
Port ['Port', 'Pont', u'Cour']
Pont ['Pont', 'Port']
Place [u'Place', 'Passage']
Passage ['Passage', u'Impasse', u'Place']
Parvis [u'Parvis']
Le ['Le']
Impasse [u'Impasse', 'Passage']
Grande ['Grande', 'Promenade']
Galerie ['Galerie']
Forum ['Forum', 'rue']
```

Correction of street names

The street names have been corrected according to the typos earlier identified.

```
rue de Tourtille ==> Rue de Tourtille
rue Auguste Beau ==> Rue Auguste Beau
rue Heloise Isabelle Michaud ==> Rue Heloise Isabelle Michaud (13 occurrences)
r Porte Jaune ==> Rue PoRuete Jaune
place de la Défense ==> Place de la Défense
rue des Trois Frères ==> Rue des Trois Frères
rue du docteur bauer ==> Rue du docteur bauer
rue de Rome ==> Rue de Rome (18 occurrences)
rue Riquet ==> Rue Riquet (2 occurrences)
villa Haussmann ==> Villa Haussmann
488 street names were fixed
```

Queries

After having converted the document into a CSV and created SQL tables, queries could have been used to obtain the following statistics about Paris:

In [190]:
```
countnodes = "SELECT COUNT(*) FROM nodes;"

for i in cur.execute(countnodes):
    print "The number of nodes is:"
    print i
```

```
The number of nodes is:
(39321,)
```

In [191]:
```
countways = "SELECT COUNT(*) FROM ways;"

for i in cur.execute(countways):
    print "The number of ways is:"
    print i
```

```
The number of ways is:
(5515,)
```

In [192]:
```
countusers = "SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM n

for i in cur.execute(countusers):
    print "The number of ways is:"
    print i
```

```
The number of ways is:
(763,)
```

In [193]:
```
# Top 5 amenities

topamenities = "SELECT value, COUNT(*) as num FROM nodes_tags WHERE

for row in cur.execute(topamenities):

    print row

```

```
('restaurant', 58)
('bench', 43)
('bicycle_parking', 36)
('parking_space', 26)
('waste_basket', 24)
```

In [196]:
```
countparking = "SELECT value, COUNT(*) FROM nodes_tags WHERE key='a

for i in cur.execute(countparking):

    print i

```

```
('parking', 7)
```

Additional ideas

It is suprirsing to notice that 776 users added information to the map. A lot of elements have been entered in the map. However, if users would have a real motive to do so, even more information could have been added. For instance, each time users would enter an element, they could get points that would be added to their score. With a high score, users could reach higher levels. Therefore, addtional data could be implemented, such as grades for the best restaurants, bars and clubs, timetable of shops and their contact.

The benefits of this new procedure are the followings:

- More data will be added (in quantity).
- Users will have to add any type of data, as long as it brings value added. More data will be added in terms of variety.

The potential problems are the followings:

- The data may lack of consistency. The more users, the more there will have different way of writing the data. For instance, some people may write postal codes with the initial of the area and some may write only the postal code numbers.
- The data may not be inaccurate. If users really want to increase their score, some may enter anything just to obtain more points.

Conclusion

As a conclusion, the data in Paris required few corrections and are in general quite consistent. This project focused on street names and amenities. Other elements could have been addressed, such as street numbers or postal codes. This project could have been improved by applying codes not only on the sample created while parsing the document, but on the entire file.

Also, the information given by Mapzen could have been more complete, such as the amount of population in certain area. So that, the concentration of amenities could have been observed according to the population density.

References

- Udacity - https://www.udacity.com/ (https://www.udacity.com/)
- Mapzen - https://mapzen.com/data/metro-extracts/ (https://mapzen.com/data/metro-extracts/)
- OpenStreetMap - https://www.openstreetmap.org (https://www.openstreetmap.org)
- Overpass API - http://overpass-api.de/ (http://overpass-api.de/)
- Python Software Foundation - https://www.python.org/ (https://www.python.org/)
- Catherine Devlin's Github repository - https://github.com/catherinedevlin/ipython-sql (https://github.com/catherinedevlin/ipython-sql)
- Python course - https://www.python-course.eu/ (https://www.python-course.eu/)