Udacity Data Wrangling Final Project

In this project, we are to select a city from https://www.openstreetmap.org and use the skills and techniques for data wrangling and cleaning to shape and audit the dataset. The data accuracy and consistensy will be assesed and adjusted as needed. Mongodb will be used to store our dataset and then queried for analytics.

The dataset

Amsterdam is the city that we will audit and clean. Amsterdam is a major city in the Netherlands and where I live. It is considered to be one of the densest cities in Europe, so it will be interesting to see how clean its data in open street map is.

Data extracted from

https://mapzen.com/data/metro-extracts/metro/amsterdam_netherlands/

Amsterdam

https://en.wikipedia.org/wiki/Amsterdam

Analysis of the Data

After inserting our cleaned data in MongoDB, we have some questions that we can answer through querying the DB. Those questions are as follows . What is the number of records? . What is the number of nodes in the data? . Who are the top 10 contributers? . What is the top amenity?

What is the number of records?

- Query used:
  - db.getCollection('open_street_map').find().count()
- Result: 2202741

What is the number of Nodes in data?

- Query used:
  - db.getCollection('open_street_map').find({"type":"node"}).count()
- Result: 2202738

Who are the top 10 contributers?

- Query used:
  - db.open_street_map.aggregate([{match: {'created.user': { exists: 1} }}, {group: {_id: 'created.user', count: {sum: 1}}}, { sort: {'count': -1}}, {$limit: 10}])
- Result:
  - { "_id" : "3dShapes", "count" : 1492463 }
  - { "_id" : "padvinder", "count" : 54286 }
  - { "_id" : "sebastic", "count" : 46598 }
  - { "_id" : "AND", "count" : 44276 }
  - { "_id" : "sebastic_BAG", "count" : 40143 }
  - { "_id" : "paulbe", "count" : 37989 }
  - { "_id" : "AnkEric", "count" : 22327 }
  - { "_id" : "Hendrikklaas", "count" : 20008 }
  - { "_id" : "Meeuw", "count" : 19828 }
  - { "_id" : "rullzer", "count" : 17603 }
- Conclusion:
  - Looks like 3dshapes is by far the most contributor. He is more than the top 9 contributors after him put together.

What is the top amenity?

- Query used:
  - db.open_street_map.aggregate([{match: {'amenity': { exists: 1} }},{group: {_id: 'amenity', count: {sum: 1}}},{ sort: {'count': -1}},{$limit: 10}])
- Result:
  - { "_id" : "recycling", "count" : 490 }
  - { "_id" : "post_box", "count" : 480 }
  - { "_id" : "parking", "count" : 320 }
  - { "_id" : "restaurant", "count" : 296 }
  - { "_id" : "bench", "count" : 257 }
  - { "_id" : "pub", "count" : 171 }
  - { "_id" : "fuel", "count" : 147 }
  - { "_id" : "fast_food", "count" : 143 }
  - { "_id" : "waste_basket", "count" : 103 }
  - { "_id" : "cafe", "count" : 97 }
- Conclusion:
  - Looks like recycling is the top amenity, followed closely by post_box. Interesting to see how few parkings there is in the data. This could be because of the limited space in amsterdam and the lack of popularity for cars.

Application Code

Here we will layout the code that will audit the data and clean it. First we will create a sample dataset from the original dataset to take less time processing it while debugging.

In [ ]:

```
import xml.etree.ElementTree as ET  # Use cElementTree or lxml if too slow

OSM_FILE = "amsterdam_netherlands.osm"  # Replace this with your osm file
SAMPLE_FILE = "sample.osm"

k = 10 # Parameter: take every k-th top level element

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

    Reference:
    http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-file-generated-via-xml-etree-elementtree-in-python
    """
    context = iter(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()
```

```
with open(SAMPLE_FILE, 'wb') as output:
    output.write(b'<?xml version="1.0" encoding="UTF-8"?>\n')
    output.write(b'<osm>\n  ')

    # Write every kth top level element
    for i, element in enumerate(get_element(OSM_FILE)):
        if i % k == 0:
            output.write(ET.tostring(element, encoding='utf-8'))

    output.write(b'</osm>')
```

Auditing street names

Street names in Dutch language did not make it easy for auditing. Unlike Enlgish, street type is part of the word. After running audit multiple times and iterating on the different type of ways, what remains are mostly street names that do not have a type attached to it. It should not be edited by us since that is a convention in Dutch culture for the street name to already contain its type. If not, it should remain the same.

Eample valide names

. Haarlemrmeerstraat --> The type is straat

. Hoofdorplein --> The type is plein

. Goude --> No type in the name but still valid

Because of this, auditing street types will not be part of the final cleaning.

In [11]:

```
import xml.etree.cElementTree as ET
import re
from collections import defaultdict
import pprint

OSMFILE = "sample.osm"
street_type_re = re.compile(r'\w*
(plein|markt|pad|dam|laan|stede|boulevard|zuid|oust|west|noord|hof|hoff|steeg|weide|kade|straat|pa
k|weg|gracht)\b', re.IGNORECASE)


def audit_street_type(street_types, street_name):
    """Update dictionary of street names and mapping type"""
    m = street_type_re.search(street_name)
    if m:
        """Value already found"""
    else:
        if street_name not in street_types:
            street_types[street_name].add(street_name)


def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")
```

Auditing Postal Codes

The accepted format for Postal codes is 4 digits followed by two letters. Some valid postal codes do not have any letters. The letters should be in captial and preferabbly no spaces. So audit will check for the format, but will automatically correct capitalization and remove spaces.

Some of the problems I found were optional postal codes seperated by ';'. For this case I simply selected the first value.

Also found cases where the postal code are for wrong cities, this cover most of the erros. All Amsterdam codes starts with 10xx while some of the codes were '2042', '1815' which lie in Zandvoort and Alkamar, cities near Amsterdam. These addresses I do not expect to get in a dataset for Amsterdam, maybe Amsterdam metropolitan area.

In [22]:

```
postal_codes = re.compile(r'^1{1}0{1}\d{2} ?([a-z]{0}|[a-z]{2})$', re.IGNORECASE)
bad_postal_codes = []
```

```
bad_postal_codes = []

def audit_postal_code(postal_code):
    postal_code = postal_code.split(';')[0].upper().replace(" ", "")
    if postal_codes.match(postal_code):
        return postal_code

    bad_postal_codes.append(postal_code)
    return None


def is_postal_code(address_key):
    return address_key == 'addr:postcode'
```

Format to JSON

We then format the data to JSON so that it can be handled easier. We also audit and transform the data while processing it.

In [24]:

```
import codecs
import json

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

CREATED = ["version", "changeset", "timestamp", "user", "uid"]
ATTRIB = ["id", "visible", "amenity", "cuisine", "name", "phone"]


def shape_element(element):
    """
    Parse, validate and format node and way xml elements.
    Return list of dictionaries
    Keyword arguments:
    element -- element object from xml element tree iterparse
    """
    if element.tag == 'node' or element.tag == 'way':

        # Add empty created dictionary and k/v = type: node/way
        node = {'created': {}, 'type': element.tag}

        # Update pos array with lat and lon
        if 'lat' in element.attrib and 'lon' in element.attrib:
            node['pos'] = [float(element.attrib['lat']), float(element.attrib['lon'])]

        # Deal with node and way attributes
        for k in element.attrib:

            if k == 'lat' or k == 'lon':
                continue
            if k in CREATED:
                node['created'][k] = element.attrib[k]
            else:
                # Add direct key/value items of node/way
                node[k] = element.attrib[k]

        # Deal with second level tag items
        for tag in element.iter('tag'):
            k = tag.attrib['k']
            v = tag.attrib['v']

            # Search for problem characters in 'k' and ignore them
            if problemchars.search(k):
                # Add to array to print out later
                continue
            elif k.startswith('addr:'):
                address = k.split(':')
                if len(address) == 2:
                    if 'address' not in node:
                        node['address'] = {}
                    if is_postal_code(k):
                        v = audit_postal_code(v)
                        if v == None:
                            return None
```

```
                    node['address'][address[1]] = v
            else:
                node[k] = v

        # Add key/value node ref from way
        node_refs = []
        for nd in element.iter('nd'):
            node_refs.append(nd.attrib['ref'])

        if len(node_refs) > 0:
            node['node_refs'] = node_refs

        return node
    else:
        return None


def process_map(file_in, pretty=False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                if pretty:
                    fo.write(json.dumps(el, indent=2) + "\n")
                else:
                    fo.write(json.dumps(el) + "\n")


process_map('sample.osm', True)

print("Total Bad postal codes: ", len(bad_postal_codes))
```

```
Total Bad postal codes:  104613
```

Conclusion

We had to ignore a large section of the downloaded Amsterdam dataset. The dataset came with a bounding box instead of an actual boundary. But once the data was cleaned to only consider amsterdam area data, it seems that the data is quite accurate and complete.

I would recommend a standardization of the postal code data from openstreetmap to just make validation easier. That includes correcting capitalization and removing spaces.

While I feel the data cleaning aspect of this was not significant enough to submit to openstreetmap due to the cleaness of the data already provided. I assume it is because of a large and active community in Amsterdam that contributed ot the data.