Wrangle OpenStreetMap Data

Overview

The open street data set chosen is from Cardiff England. The following scripts will perform data cleaning, auditing and wrangling. First, mistakes and abbreviation will be cross validated with Google Map. The cleaned data set will then be stored in SQL database. A few statistical results will be provided at the end of the report.

Data Cleaning and Auditing

The trial cleaning is in the "Open_Street_Map.py". I was amazed by the different street types and names in Cardiff, for instance, "Heol Eglwys", "Heol-Y-Deri", and "Pen-Y-Bryn". There weren't many abbreviations in street names; however, there are a few mistakes in the file.

1. There is a mis-spelling of "Cresent", which is supposed to be "Crescent"
2. Some "Avenue", "Street", "Road" are written in all small letters
3. "Pierhead Street" is written as "Pierheadstreet" by mistake
4. The content of "v" tag is sometimes the full address not the street name
5. Some street names are not capitalized properly

For problems 1, 2 and 3, I used the mapping function to map the wrong character to the correct one. The python script I used to achieve this is shown below.

```
mapping = { "Garden": "Gardens",
        "Cresent": "Crescent",
        "Pierheadstreet": "Pierhead Street",
        "W" : "West",
        "Rd" : "Road"}

def update_name(name, mapping):
    for target in mapping:
        if target in name:
            return name.replace(target, mapping[target])
    return name
```

For problem 4 and 5, I first split the street data by ",", and take the first part of the value as the street name. Later, I use "title()" to capitalize the first letter of every word. However, the "title()" will also wrongly capitalize the "s" after "'s". Then I replaced the "'S" with "'s" to ensure all the words are in the right case. Then, I reduced these problems to problem 1 and 2, so I call the above functions again to update the names. The code that achieves what I mentioned is

```
def audit_street_type(street_types, street_name):
    search_result = street_type_re.search(street_name)
    if search_result is not None:
        search_type = search_result.group().title()
        if search_type not in expected:
            street_name = update_name(street_name, mapping)
            search_result = street_type_re.search(street_name)
            search_type = search_result.group().title()
            if search_type not in expected:
                street_types[search_type].add(street_name)
        else:
            street_types[search_type].add(street_name)
```

Other Problems in StreetName

If one takes a closer look in the data set, one may notice that some street names are wrong. For instance, "Browing Street" is supposed to be "Browning Street". However, since I do not have a dictionary for all the possible spelling of the OpenStreet Data, it is almost impossible to eliminate all the spelling mistakes in the file. I could use some tools from NLTK library to iterate through all the street names and pick out the suspicious ones for further examination. Since it' s beyond the purpose of this project, I may add this part in in the future.

Data Validation and Import

In "validation_and_import.py" file, I've implemented another program that execute the same logic as shown above but will only modify one street name a time. The open street data set is split into nodes_tags.csv, nodes.csv, ways.csv, ways_tags.csv and ways_nodes.csv, where a node contains the location information, and tags that record functional or instructional information. Similarly, a way contains reference to its nodes and relevant tags.

Overview Statistics

Findally, the script used to execute SQL command is in "Statistics and obervation.py"
This section provides an overview of the exported file and some insights about the data.

File Sizes

Files	Size

node_tags.csv	500KB
nodes	28.5MB
ways_nodes	11.2MB
ways_tags	3.9MB
ways	2.9MB
cardiff.osm	76.2MB
Open_Street_Data.db	42.3MB

Primary Research

```
def primary_research(conn):
    cur = conn.cursor()
    cur.execute('''SELECT COUNT(*) FROM nodes;''')
    rows = cur.fetchall()
    for row in rows:
        print "The number of nodes is " + str(row[0])

    cur.execute ('''SELECT COUNT(*) FROM ways;''')
    rows = cur.fetchall()
    for row in rows:
        print "The number of ways is " + str(row[0])

    cur.execute('''SELECT COUNT(DISTINCT(total.uid)) FROM
    (SELECT uid FROM nodes
    UNION ALL
    SELECT uid FROM ways) AS total;''')
    rows = cur.fetchall()
    for row in rows:
        print "The number of unique user is " + str(row[0])

    cur.execute('''SELECT  COUNT(value) FROM nodes_tags
    WHERE key = "amenity"
    AND value = "cafe";''')
    rows = cur.fetchall()
    for row in rows:
        print "The number of cafe is " + str(row[0])
```

After running this function, the result gives:

```
1. Conducting Primary research
```

```
The number of nodes is 346300
The number of ways is 49066
The number of unique user is 552
The number of cafe is 99
```

Finding Accessible Road

Ways_tags include tags that indicate whether a road is accessible for wheelchair users, in order to find those ways, I wrote the following script

```
def accessible_road(conn, selection):
    accessible_road =[]
    cur = conn.cursor()
    if selection == "yes":
        cur.execute('''SELECT  DISTINCT ways_tags.value FROM
        (SELECT ways_tags.id FROM ways_tags
        WHERE  ways_tags.key = "wheelchair"
        AND ways_tags.value = "yes") as Result
        JOIN ways_tags
        ON ways_tags.id = Result.id
        WHERE ways_tags.key = "name";''')

    if selection == "limited":
        cur.execute('''SELECT  DISTINCT ways_tags.value FROM
        (SELECT ways_tags.id FROM ways_tags
        WHERE  ways_tags.key = "wheelchair"
        AND ways_tags.value = "limited") as Result
        JOIN ways_tags
        ON ways_tags.id = Result.id
        WHERE ways_tags.key = "name";''')

    if selection == "all":
        cur.execute('''SELECT  DISTINCT ways_tags.value FROM
        (SELECT ways_tags.id FROM ways_tags
        WHERE  ways_tags.key = "wheelchair"
        AND (ways_tags.value = "limited"
        OR ways_tags.value = "yes")) as Result
        JOIN ways_tags
        ON ways_tags.id = Result.id
        WHERE ways_tags.key = "name";''')

    rows = cur.fetchall()
    ind = 1
    for row in rows:
        accessible_road.append((ind, row[0].encode("utf-8")))
        ind +=1
    print accessible_road
```

The script takes in two parameters, one can specify whether full accessibility or/and limited accessibility needed. The first 10 result follows

```
3. Find all accessible places
Find all places with accessiblitiy support (labeled 'accssibility-yes')

[(1, 'Principality Stadium'),
(2, 'Amgueddfa Genedlaethol Caerdydd / National Museum Cardiff'),
(3, 'Tesco'),
(4, 'Tesco Extra Cardiff'),
(5, "Queen's Arcade"),
(6, "Saint David's Centre"),
(7, 'Capitol'),
(8, 'Cardiff Central Library'),
(9, 'Cardiff Metropolitan University, Llandaff Campus'),
(10, 'John Lewis')]

Find all places with limited accessibility support
(labeled 'accssibility-limited')
[(1, 'Cardiff City Hall'), (2, 'Cardiff Royal Infirmary')]

Find all places with any kind of accessibility support
(labeled 'accessibiltiy-limited and labeled 'accessibility-all')
[(1, 'Principality Stadium'),
(2, 'Cardiff City Hall'),
(3, 'Amgueddfa Genedlaethol Caerdydd / National Museum Cardiff'),
(4, 'Tesco'),
(5, 'Tesco Extra Cardiff'),
(6, "Queen's Arcade"),
(7, "Saint David's Centre"),
(8, 'Capitol'),
(9, 'Cardiff Central Library'),
(10, 'Cardiff Metropolitan University, Llandaff Campus')]
------------------------------
```

Favourite Cuisine

In Cardiff, fish_and_chips is a type of cuisine that is specially listed other than the usual cuisine categories, like Chinese, Japanese and American, in North America. Not surprisingly, fish_and_chips outnumbered all other cuisine types in Cardiff.

```
def find_most_popular_cuisines(conn):
    cur = conn.cursor()
    cur.execute('''SELECT value, COUNT(*) as cuisine_count FROM nodes_tags
    WHERE key="cuisine"
    GROUP BY value
    ORDER BY cuisine_count DESC
    LIMIT 10;''')
    rows = cur.fetchall()
    for row in rows:
        cuisine = row[0].encode("utf-8")
        print cuisine + (20-len(cuisine))*" "+ "|" + str(row[1])
```

It gives

```
2. Find the most favourite cuisine, my guess is fish'n'chips...
fish_and_chips      |22
chinese             |21
indian              |21
coffee_shop         |16
italian             |12
pizza               |11
sandwich            |8
burger              |5
chicken             |4
japanese            |4
```

However, when I printed the least popular cuisines, I noticed

```
Lebanese,_Middle_Eastern|1
Menu:_seasonscardiff.co.uk|1
Persian             |1
brezilian           |1
chinese;fish_and_chips|1
cornish_pasty       |1
crepe               |1
fish                |1
fish_and_chips;indian|1
fish_and_chips;pizza;kebab;burger|1
```

where some cuisines are a combination of other cuisines. Then I decided to find the total number of the cuisines that have fish_and_chips. The script is shown below

```
def find_all_fish_and_chips(conn):
    cur = conn.cursor()
    cur.execute('''SELECT COUNT(value) FROM nodes_tags
    WHERE key = "cuisine"
    AND value LIKE "%fish_and_chips%";''')
    rows = cur.fetchall()
    for row in rows:
        print row[0]
```

which gives

```
After cleaning, the total fish_and_chips cuisine are
25
```

Audit Postal Code

After I have audited the postal code data, I found that postal codes are stored in either "postal_code" or "postcode". After I ran the following script

```
def audit_cardiff_postal_code(conn):
    cur = conn.cursor()
    cur.execute('''SELECT ways_tags.key, ways_tags.value FROM ways_tags
    JOIN (SELECT audit.value, audit.id
        FROM (SELECT * FROM nodes_tags
              UNION ALL
            SELECT * FROM ways_tags) as audit
        WHERE (audit.key='postcode'
        OR audit.key = "postal_code")
        AND NOT audit.value LIKE "CF%"
        GROUP BY audit.value) as invalid
        ON invalid.id = ways_tags.id''')
    rows = cur.fetchall()
    print [(row[0].encode('utf-8'), row[1].encode('utf-8')) for row in rows]
```

I noticed one entry that has postal code that does not give a proper format.

```
[('name', 'Roxby Court'),
('highway', 'residential'),
('postal_code', 'CF10 4AG'),
('postcode', 'OSM Note by HimanshuS')]
```

The entry has both postal_code and postcode field and the postal code information is stored under "postal_code" field
I started to wonder if other entries have both "postal_code" and "post_code" fields and stored replicated information,
then I ran the following script

```
def find_entries_have_both_postal_and_postcode(conn):
    cur = conn.cursor()
    cur.execute('''SELECT audit.value, audit.id, audit.key
    FROM (SELECT * FROM nodes_tags
    UNION ALL
    SELECT * FROM ways_tags) as audit
```

```
      jOIN (SELECT * FROM nodes_tags
      UNION ALL
      SELECT * FROM ways_tags) as audit_copy
      ON audit.id = audit_copy.id
      WHERE audit.id = audit_copy.id
      AND (audit.key = "postal_code"
      AND audit_copy.key = "postcode")
      OR (audit_copy.key = "postal_code"
      AND audit.key = "postcode");''')
      rows = cur.fetchall()
      print [(row[0].encode('utf-8'), row[1], row[2].encode('utf-8'))\
       for row in rows]
```

The result shows that the only entry has both fields is the entry I mentioned above

```
[('CF10 4AG', 477291415, 'postal_code'),
('OSM Note by HimanshuS', 477291415, 'postcode')]
```

Conclusion

After reviewing this data, it is quite clear that some values of the ways and node tags are not stored uniformly. For instance, some cuisines are stored as single values, but some have multiple values. Also, the multi-value fields are sometimes split by ";" but "," other times. Also, "postal_code" and "postcode" are used interchangeably. One user filled both "postal_code" and "postcode", but only one field contains the correct field. There are also some mis-spelling problems in some of the values that requires update.

Overall, the OpenStreet Data in Cardiff provides an overview of the roads and stores information which can be convenient for people to look up places for leisure activities and also for people with accessibility needs. However, if there is a validator or rules to validate the entries from users, the data can be "cleaner" and information can be more retrievable.