OpenStreetMap Data Wrangling with SQL

1 Map Area

Colchester, England

- https://www.openstreetmap.org/relation/76493#map=11/51.8776/0.9033
- http://metro.teczno.com/#colchester

I chose this particular area because it is the where I have lived in the recent years, I know it well and would like to help improve it if possible.

2 Problems Encountered in the Map

- Unexpected "k" values for <tag>
- Inconsistent street names & different expected street types between America and England
- Abnormal Post Codes

2.1 Unexpected "k" Values for <tag>

Within **tags.py** and **data.py**, regular expressions are used to check the **"k"** value for each **<tag>** . The **data.py** has been changed to check the type of tags and correct the type, key, and value of **<key>**, and was exported to **nodes_tags**, **ways_tags.csv** files.

Tthe logic of auditing **"k"** value is as below:

- Firstly, if the "k" value matches the **PROBLEMCHARS**, then I ignored it and it wouldn't be saved to the .csv file;
- Secondly, if the **"k"** value is a postcode key, then update it to the standard k value for postcode, which is "**addr:postcode**", I checked this particularly because i noticed the postcode **"k"** values vary, generally it is "**addr:postcode**", but sometimes it is "**postcal_code**" or "**postcode**".
- Thirdly, if the **"k"** value contains colon **(:)**, then split the k value to get tag type and tag key.
- Finally, set the tag type as default tag type and tag key as the current **"k"** value.

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

```
def is_postcode_key(k_name):
    return (k_name == "addr:postcode") or (k_name == "postal_code") or (k_name == "postcode")

def get_tag_dict(id, tag, problem_chars=PROBLEMCHARS, default_tag_type='regular'):
    tag_dict = {}
    tag_dict['id'] = id
    tag_dict['value'] = tag.attrib["v"]
    cur_k = tag.attrib["k"]
    if problem_chars.search(cur_k) is not None:
        return None
    elif is_postcode_key(cur_k):
        cur_k = "addr:postcode"
        tag_dict["value"] = update_postcode(tag.attrib["v"])
        if tag_dict["value"] is not None:
            cur_k_split = cur_k.split(":", 1 )
            tag_dict["type"] = cur_k_split[0]
            tag_dict["key"] = cur_k_split[1]
        else:
            return None
    elif LOWER_UPPER_COLON.search(cur_k) is not None:
        if cur_k == "addr:street":
            tag_dict["value"] = audit_and_update_street_name(tag.attrib["v"])
        cur_k_split = cur_k.split(":", 1 )
        tag_dict["type"] = cur_k_split[0]
        tag_dict["key"] = cur_k_split[1]
    else:
        tag_dict["type"] = default_tag_type
        tag_dict["key"] = cur_k
    return tag_dict
```

2.2 Incorrect Postcodes

Another script **audit_postcode.py** is used to check if the postcodes is correct. A regular expression is written according to the postcode regulation in Colchester, England

```
postcode_pattern = re.compile(r'^[A-Z]{2}[1-9][0-9]?\s[0-9]{1,2}[A-Z]{2}$')
```

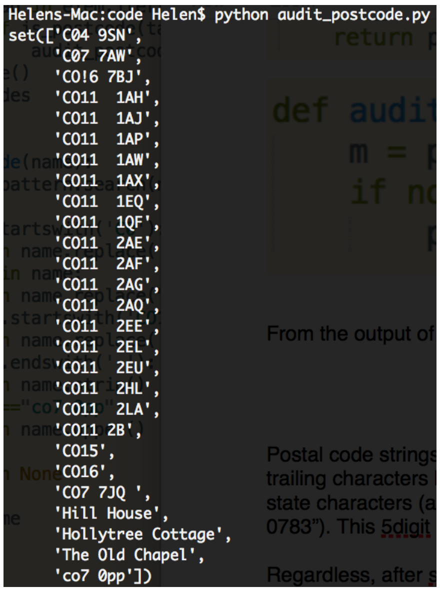
```python
def audit(osmfile):
    osm_file = open(osmfile, "r")
    postcodes = set()
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_postcode(tag):
                    audit_postcode(postcodes, tag.attrib['v'])
    osm_file.close()
    return postcodes
```

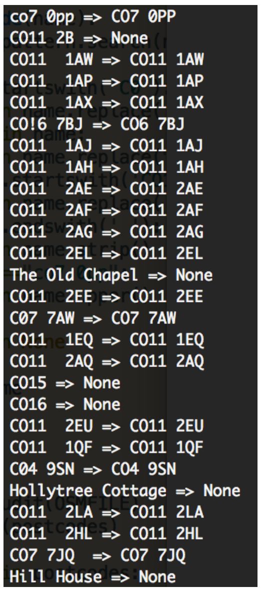
```python
def is_postcode(elem):
    return (elem.attrib['k'] == "addr:postcode") or \
        (elem.attrib['k'] == "postal_code") or \
        (elem.attrib['k'] == "postcode")
```

```python
def audit_postcode(postcodes, cur_postcode):
    m = postcode_pattern.search(cur_postcode)
    if not m:
        postcodes.add(cur_postcode)
```

From the output of the functions above, I found there were some incorrect postcodes which needed to be corrected.



```
Helens-Mac:code Helen$ python audit_postcode.py
set(['C04 9SN',
     'C07 7AW',
     'C0!6 7BJ',
     'C011  1AH',
     'C011  1AJ',
     'C011  1AP',
     'C011  1AW',
     'C011  1AX',
     'C011  1EQ',
     'C011  1QF',
     'C011  2AE',
     'C011  2AF',
     'C011  2AG',
     'C011  2AQ',
     'C011  2EE',
     'C011  2EL',
     'C011  2EU',
     'C011  2HL',
     'C011  2LA',
     'C011 2B',
     'C015',
     'C016',
     'C07 7JQ ',
     'Hill House',
     'Hollytree Cottage',
     'The Old Chapel',
     'co7 0pp'])
```



```
co7 0pp => C07 0PP
C011 2B => None
C011  1AW => C011 1AW
C011  1AP => C011 1AP
C011  1AX => C011 1AX
C0!6 7BJ => C06 7BJ
C011  1AJ => C011 1AJ
C011  1AH => C011 1AH
C011  2AE => C011 2AE
C011  2AF => C011 2AF
C011  2AG => C011 2AG
C011  2EL => C011 2EL
The Old Chapel => None
C011  2EE => C011 2EE
C07 7AW => C07 7AW
C011  1EQ => C011 1EQ
C011  2AQ => C011 2AQ
C015 => None
C016 => None
C011  2EU => C011 2EU
C011  1QF => C011 1QF
C04 9SN => C04 9SN
Hollytree Cottage => None
C011  2LA => C011 2LA
C011  2HL => C011 2HL
C07 7JQ  => C07 7JQ
Hill House => None
```

```python
def update_postcode(name):
    m = postcode_pattern.search(name)
    if not m:
        if name.startswith('C0'):
            return name.replace('C0',"CO")
        elif "!" in name:
            return name.replace("!","")
        elif name.startswith('C011  '):
            return name.replace('C011  ',"C011 ")
        elif name.endswith(' '):
            return name.strip()
        elif name=="co7 0pp":
            return name.upper()
        else:
            return None
    else:
        return name
```

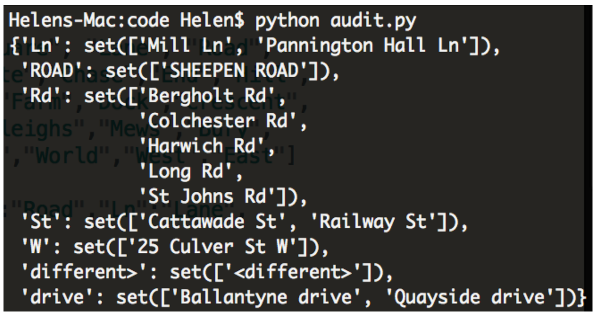
2.3 Abnormal Street Names

The script **audit.py** is used to check the street names, the problem encountered is there are quite a lot of street types which are not in the expected type list, but actually they are normal and correct street types, such as Chase, Close, Crescent, Cross, Grove etc,. Probably this is because of the culture difference between England and America. I updated the expected street type list to add the normal street types, and the mapping list was also updated according to the specific situation.

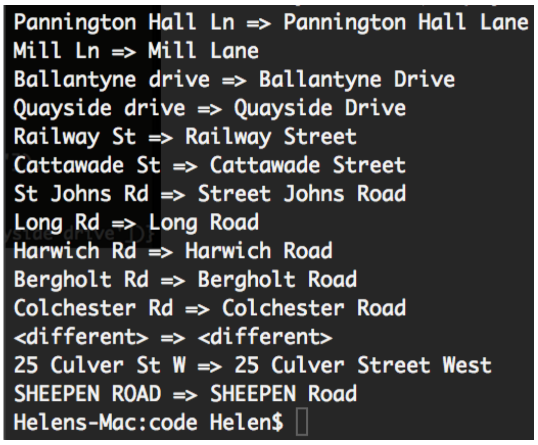
```python
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons","Way","Close","Grove","Cross","Gate","Chase","End","Hill",
            "Park","Ravensdale","Newlands","Waterside","Robinsdale","Quay","Farm","Dock","Crescent",
            "Gardens","Walk","Park","Mount","View","Parade","Headgate","Rayleighs","Mews","Bury",
            "Marina","Green","Meadway","Vale","Rise","Middleborough","Heath","World","West","East"]

mapping = { "St": "Street", "St.": "Street","Rd.":"Road","Rd":"Road","ROAD":"Road","Ln":"Lane",
            "Ave":"Avenue","drive":"Drive","W":"West"}
```

```python
def update_name(name, mapping):
    new_street_list = []
    for piece in name.split(" "):
        if piece in mapping.keys():
            piece = mapping[piece]
        new_street_list.append(piece)
    return " ".join(new_street_list)
```



```
Helens-Mac:code Helen$ python audit.py
{'Ln': set(['Mill Ln', 'Pannington Hall Ln']),
 'ROAD': set(['SHEEPEN ROAD']),
 'Rd': set(['Bergholt Rd',
            'Colchester Rd',
            'Harwich Rd',
            'Long Rd',
            'St Johns Rd']),
 'St': set(['Cattawade St', 'Railway St']),
 'W': set(['25 Culver St W']),
 'different>': set(['<different>']),
 'drive': set(['Ballantyne drive', 'Quayside drive'])}
```

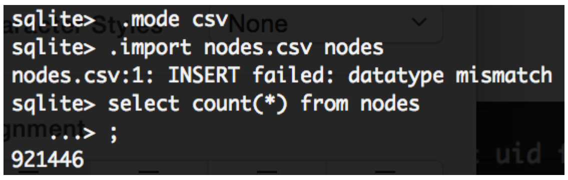


```
Pannington Hall Ln => Pannington Hall Lane
Mill Ln => Mill Lane
Ballantyne drive => Ballantyne Drive
Quayside drive => Quayside Drive
Railway St => Railway Street
Cattawade St => Cattawade Street
St Johns Rd => Street Johns Road
Long Rd => Long Road
Harwich Rd => Harwich Road
Bergholt Rd => Bergholt Road
Colchester Rd => Colchester Road
<different> => <different>
25 Culver St W => 25 Culver Street West
SHEEPEN ROAD => SHEEPEN Road
Helens-Mac:code Helen$
```

3 Overview of the Data

This section contains basic statistics about the dataset and the SQL queries used to gather them.

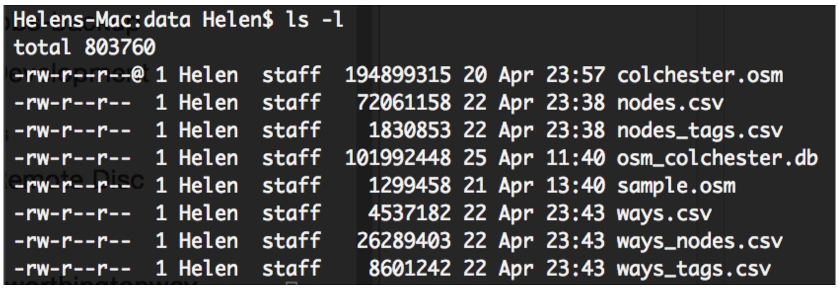
First of all, after all the **.csv** files were exported from **data.py**, using the following commands to import the cvs files to the sqlite database, which was created according to the schema provided.



```
sqlite> .mode csv
sqlite> .import nodes.csv nodes
nodes.csv:1: INSERT failed: datatype mismatch
sqlite> select count(*) from nodes
   ...> ;
921446
```

Note: "**the INSERT failed: datatype mismatch**" is the error when the first row is the header rather than data, the rest of the file was imported correctly.

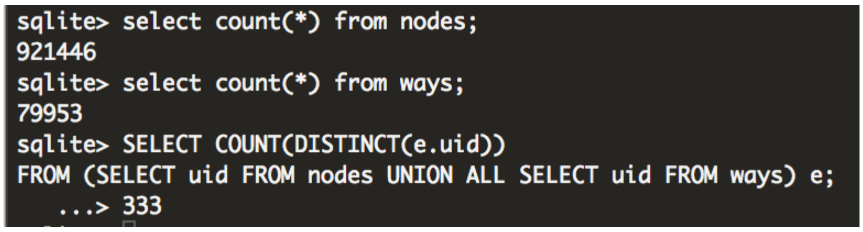
3.1 File sizes



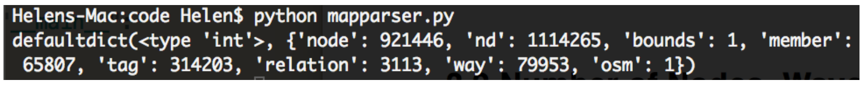
```
Helens-Mac:data Helen$ ls -l
total 803760
-rw-r--r--@ 1 Helen  staff  194899315 20 Apr 23:57 colchester.osm
-rw-r--r--  1 Helen  staff   72061158 22 Apr 23:38 nodes.csv
-rw-r--r--  1 Helen  staff    1830853 22 Apr 23:38 nodes_tags.csv
-rw-r--r--  1 Helen  staff  101992448 25 Apr 11:40 osm_colchester.db
-rw-r--r--  1 Helen  staff    1299458 21 Apr 13:40 sample.osm
-rw-r--r--  1 Helen  staff    4537182 22 Apr 23:43 ways.csv
-rw-r--r--  1 Helen  staff   26289403 22 Apr 23:43 ways_nodes.csv
-rw-r--r--  1 Helen  staff    8601242 22 Apr 23:43 ways_tags.csv
```

- The dataset **colchester.osm** is 194M
- The database file is 102M
- The sample dataset is 1.3M

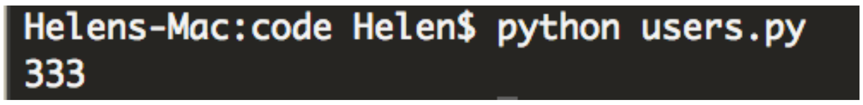
3.2 Number of Nodes, Ways, Unique Users



```
sqlite> select count(*) from nodes;
921446
sqlite> select count(*) from ways;
79953
sqlite> SELECT COUNT(DISTINCT(e.uid))
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;
   ...> 333
```



```
Helens-Mac:code Helen$ python mapparser.py
defaultdict(<type 'int'>, {'node': 921446, 'nd': 1114265, 'bounds': 1, 'member':
 65807, 'tag': 314203, 'relation': 3113, 'way': 79953, 'osm': 1})
```

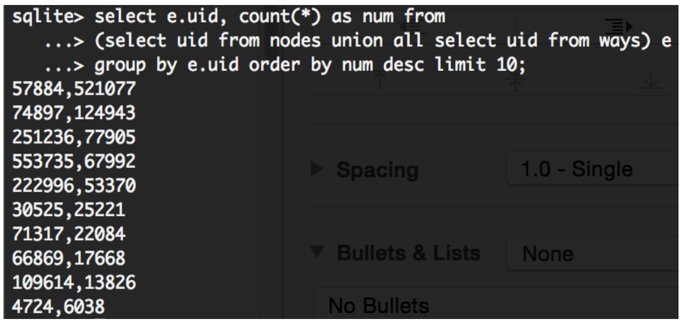


```
Helens-Mac:code Helen$ python users.py
333
```

The script **mapparser.py** was used to count occurrences of each tag, we can see the results of SQL and python match.
Also, the script **users.py** is used to count how many unique users have contributed to the map in Colchester, England, the running result is also 333 unique users.

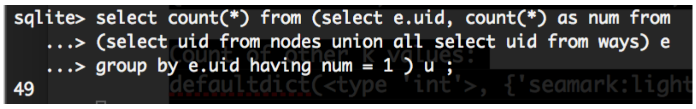
3.3 Users' Contribution

Top 10 contributing users:



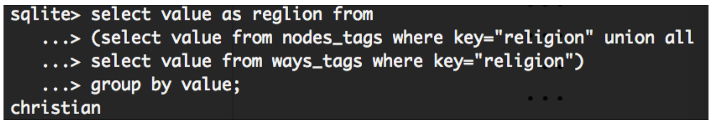
```
sqlite> select e.uid, count(*) as num from
   ...> (select uid from nodes union all select uid from ways) e
   ...> group by e.uid order by num desc limit 10;
57884,521077
74897,124943
251236,77905
553735,67992
222996,53370
30525,25221
71317,22084
66869,17668
109614,13826
4724,6038
```

Number of users having only 1 post:

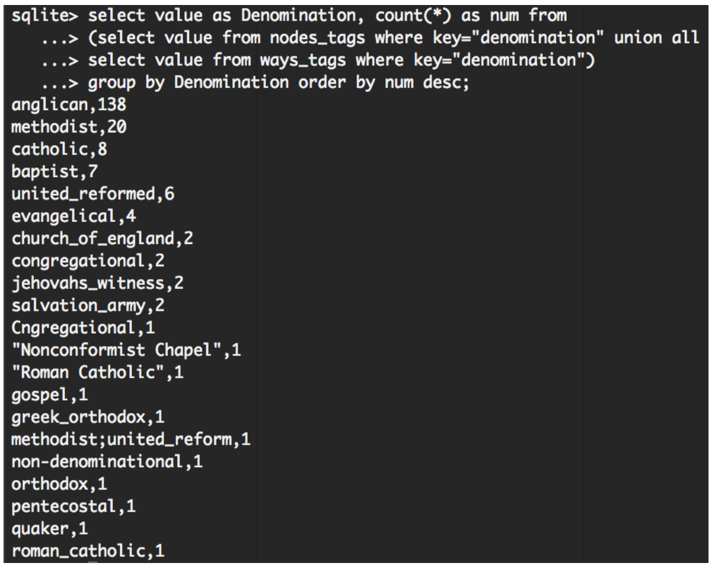


```
sqlite> select count(*) from (select e.uid, count(*) as num from
   ...> (select uid from nodes union all select uid from ways) e
   ...> group by e.uid having num = 1 ) u ;
49
```

3.4 Region & Denomination



```
sqlite> select value as reglion from
   ...> (select value from nodes_tags where key="religion" union all
   ...> select value from ways_tags where key="religion")
   ...> group by value;
christian
```

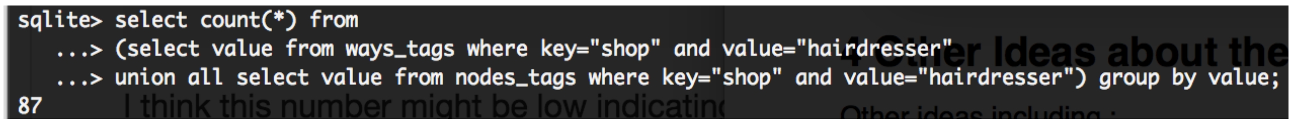


```
sqlite> select value as Denomination, count(*) as num from
   ...> (select value from nodes_tags where key="denomination" union all
   ...> select value from ways_tags where key="denomination")
   ...> group by Denomination order by num desc;
anglican,138
methodist,20
catholic,8
baptist,7
united_reformed,6
evangelical,4
church_of_england,2
congregational,2
jehovahs_witness,2
salvation_army,2
Cngregational,1
"Nonconformist Chapel",1
"Roman Catholic",1
gospel,1
greek_orthodox,1
methodist;united_reform,1
non-denominational,1
orthodox,1
pentecostal,1
quaker,1
roman_catholic,1
```

With the SQL queries on the left, it is clear to see that the religion in Colchester is **christian**. And the most common denomination is **anglican**, followed by **methodist**, **catholic**, and **baptist**.

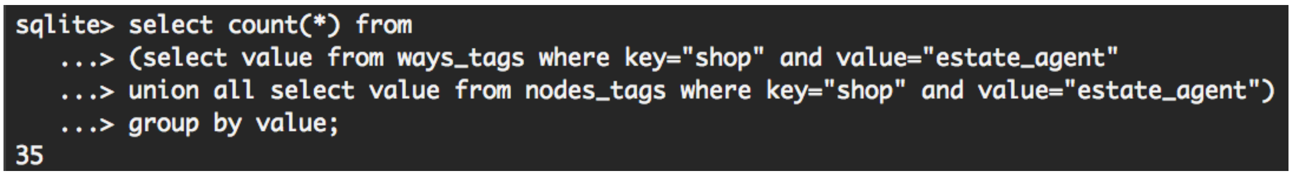
3.5 Number of Chosen Type of Nodes

There are 87 hairdresser shops in Colchester.



```
sqlite> select count(*) from
   ...> (select value from ways_tags where key="shop" and value="hairdresser"
   ...> union all select value from nodes_tags where key="shop" and value="hairdresser") group by value;
87
```

There are 35 estate agents in Colchester:



```
sqlite> select count(*) from
   ...> (select value from ways_tags where key="shop" and value="estate_agent"
   ...> union all select value from nodes_tags where key="shop" and value="estate_agent")
   ...> group by value;
35
```

4 Other Ideas and Comments

- More than 50% of nodes don't have postcode information. It might be helpful to improve this because postcode is more precise and helpful than street names.

- After querying from the database with "*amenity*" as **key** and "*school*" as **value**, I found that only school name is saved with the **key** "*name*". While when the amenity is a restaurant, more details such as website, cuisine, phone number, etc. were saved in the tags. It might be helpful if more details for schools can be saved in the tags as well, such as school website, Ofsted rating, age range of pupils etc.

4.1 Improve Missing Postcode

This improvement can greatly make the node/way location more precise in location because as I know the street names in Colchester are not unique. Sometimes the street names are exactly the same even when the two roads are quite close to each other.

However, implementing this improvement can be very tedious and time-consuming. One possible solution I can imagine is getting the missing postcodes from Google Maps API because each node has longitude and latitude information.

4.2 More Detailed Tags '*<tag>*' for '*amenity:school*'

This improvement will greatly benefit the parents who are looking for a school for their children. Information such as Ofsted rating, age range of pupils, religion etc. Another related improvement can be adding a tag for a way/node to show which school's priority admission area it is located in.

Again, this improvement is not easy to implement as information is not easy to get automatically and sometimes the content of the information is not organised and structured. One possible solution I can think of is to get data from the Colchester Council website as they surely store the detailed information for schools also schools' priority admission area. However, one concern is using information from this website seems like data stealing and may violate the terms of agreement for Colchester Council.

4.3 Final Note

Note: There are still several opportunities for cleaning and validation that I left unexplored. However, I believe the dataset was well-cleaned for the purposes of this project.