OpenStreetMap Data Wrangling for Melbourne

The area of the map
Melbourne, Australia

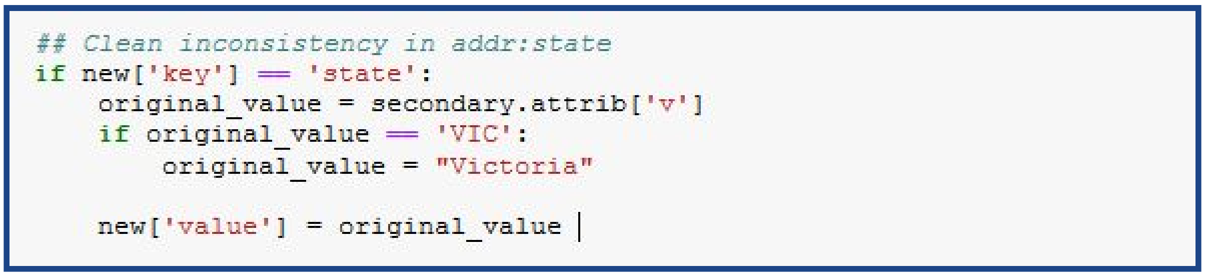
This map is of Melbourne city, Australia. I went to choose this area of the world as i have spent 4 years of my life on it and always want to know more about his land. I want to explore this map and dataset to unleash the information about the city through some queries and cleaning processes.

Some Problems Found in this map

- The “addr:state” values are written in two different ways (VIC, Victoria)
- There is no consistency in source keys; same values are written in different cases (upper or lower case).
- There is an inconsistency in phone number formats (some include whitespaces while others don’t, some include area code ’613’ while others do not)

The inconsistency in state values

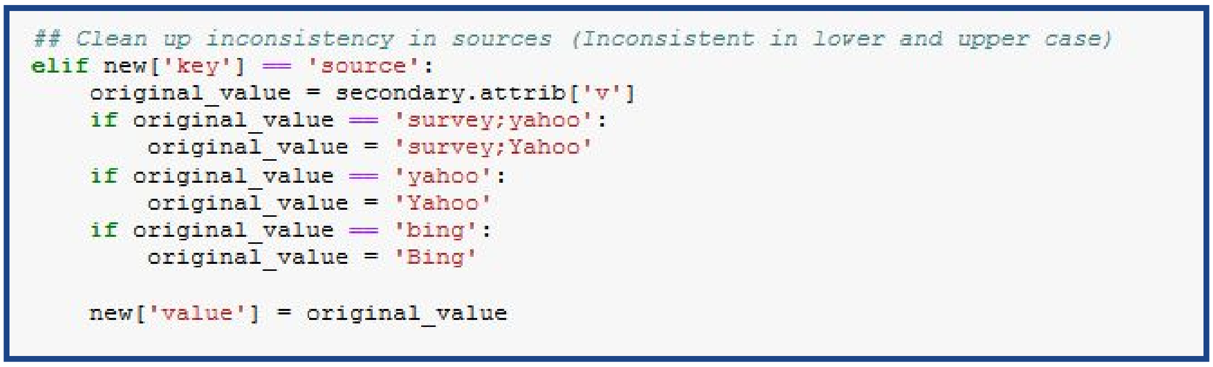
The “addr:state” key as we also saw in the lesson is common in having inconsistencies. For this dataset one method of code to solve the problem. The state values were written as “VIC” or “Victoria”, changing all values to just “Victoria” makes it easier to read and makes the data more solid and consistent.



```
## Clean inconsistency in addr:state
if new['key'] == 'state':
    original_value = secondary.attrib['v']
    if original_value == 'VIC':
        original_value = "Victoria"

    new['value'] = original_value
```

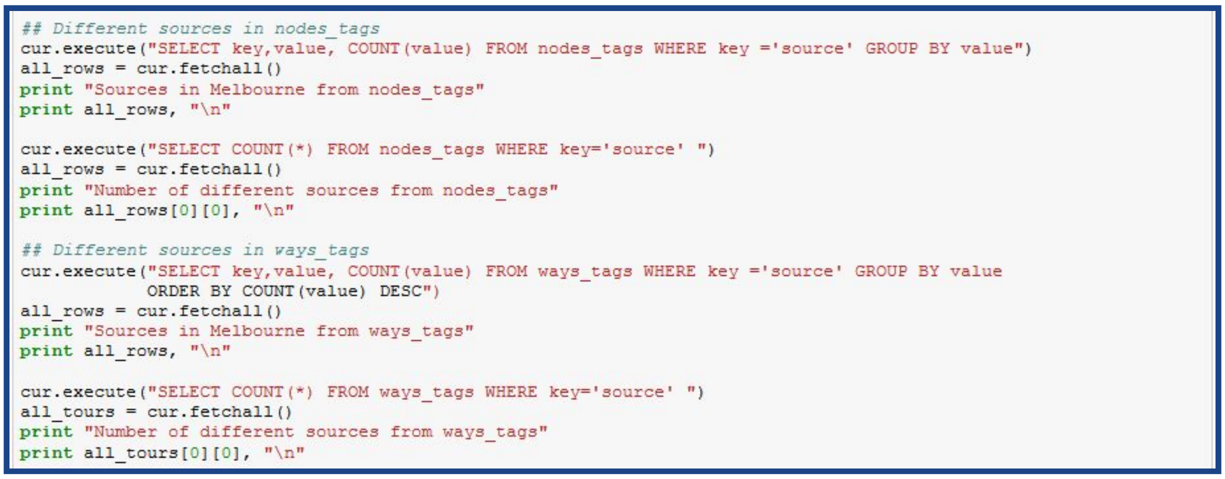
The inconsistency in source keys

The source key values in both nodes_tags and ways_tags had some inconsistency. Same values were written in different ways, such as one in uppercase and the exact same word in lowercase somewhere else. An issue may occur when dealing with large numbers of data, so it is better to put all words that have the same meaning in one format.



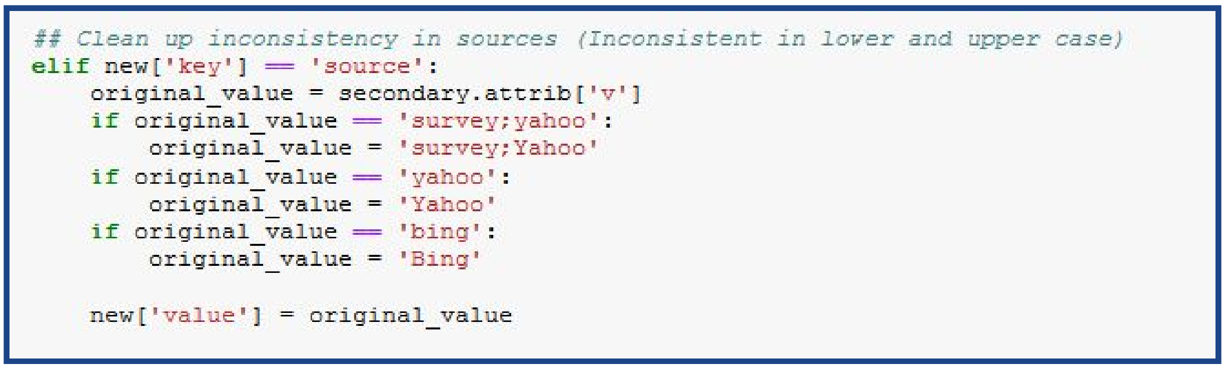
```
## Clean up inconsistency in sources (Inconsistent in lower and upper case)
elif new['key'] == 'source':
    original_value = secondary.attrib['v']
    if original_value == 'survey;yahoo':
        original_value = 'survey;Yahoo'
    if original_value == 'yahoo':
        original_value = 'Yahoo'
    if original_value == 'bing':
        original_value = 'Bing'

    new['value'] = original_value
```

When exporting the data to the database from the csv files, some querying faced this issue.



```
## Different sources in nodes_tags
cur.execute("SELECT key,value, COUNT(value) FROM nodes_tags WHERE key ='source' GROUP BY value")
all_rows = cur.fetchall()
print "Sources in Melbourne from nodes_tags"
print all_rows, "\n"

cur.execute("SELECT COUNT(*) FROM nodes_tags WHERE key='source' ")
all_rows = cur.fetchall()
print "Number of different sources from nodes_tags"
print all_rows[0][0], "\n"

## Different sources in ways_tags
cur.execute("SELECT key,value, COUNT(value) FROM ways_tags WHERE key ='source' GROUP BY value
            ORDER BY COUNT(value) DESC")
all_rows = cur.fetchall()
print "Sources in Melbourne from ways_tags"
print all_rows, "\n"

cur.execute("SELECT COUNT(*) FROM ways_tags WHERE key='source' ")
all_tours = cur.fetchall()
print "Number of different sources from ways_tags"
print all_tours[0][0], "\n"
```

Most of the cleaning process takes place in the clean_tag() function.



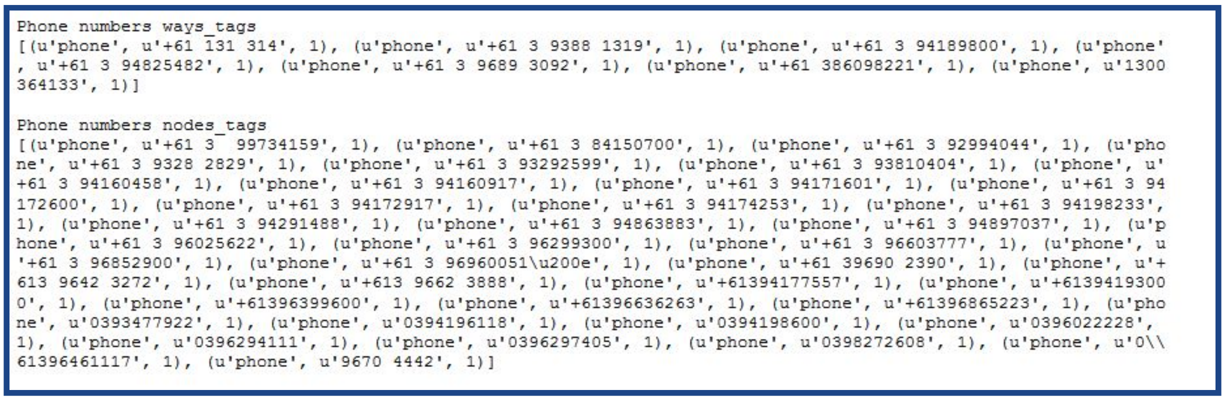
```
## Clean up inconsistency in sources (Inconsistent in lower and upper case)
elif new['key'] == 'source':
    original_value = secondary.attrib['v']
    if original_value == 'survey;yahoo':
        original_value = 'survey;Yahoo'
    if original_value == 'yahoo':
        original_value = 'Yahoo'
    if original_value == 'bing':
        original_value = 'Bing'

    new['value'] = original_value
```

A variable called "new" is created which is a dictionary for storing all the new versions of the data that is to be cleaned. The original value from the xml file is first assigned to the 'value' key in the new dictionary variable.

Then there are 'if' statements to check whether that original variable should stay the way it is or if it requires adjustments. If it needs to be adjusted, then the original_value variable is assigned the new value and changed in "new" (the dictionary).

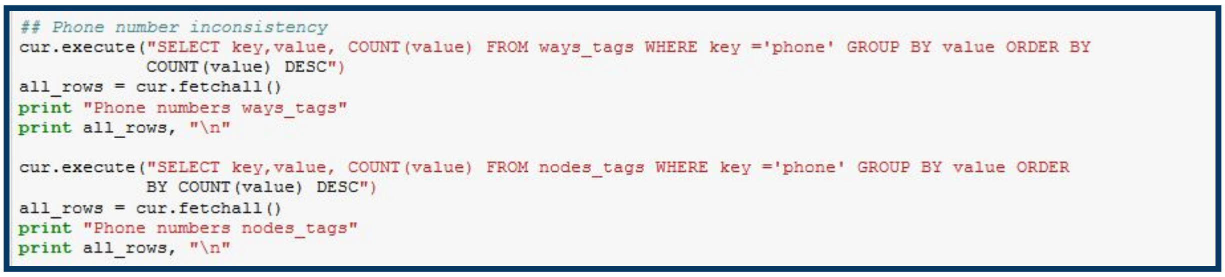
The clean_tag() function can also be viewed the update function.

Inconsistency in phone numbers



```
Phone numbers ways_tags
[(u'phone', u'+61 131 314', 1), (u'phone', u'+61 3 9388 1319', 1), (u'phone', u'+61 3 94189800', 1), (u'phone'
, u'+61 3 94825482', 1), (u'phone', u'+61 3 9689 3092', 1), (u'phone', u'+61 386098221', 1), (u'phone', u'1300
364133', 1)]

Phone numbers nodes_tags
[(u'phone', u'+61 3  99734159', 1), (u'phone', u'+61 3 84150700', 1), (u'phone', u'+61 3 92994044', 1), (u'pho
ne', u'+61 3 9328 2829', 1), (u'phone', u'+61 3 93292599', 1), (u'phone', u'+61 3 93810404', 1), (u'phone', u'
+61 3 94160458', 1), (u'phone', u'+61 3 94160917', 1), (u'phone', u'+61 3 94171601', 1), (u'phone', u'+61 3 94
172600', 1), (u'phone', u'+61 3 94172917', 1), (u'phone', u'+61 3 94174253', 1), (u'phone', u'+61 3 94198233',
1), (u'phone', u'+61 3 94291488', 1), (u'phone', u'+61 3 94863883', 1), (u'phone', u'+61 3 94897037', 1), (u'p
hone', u'+61 3 96025622', 1), (u'phone', u'+61 3 96299300', 1), (u'phone', u'+61 3 96603777', 1), (u'phone', u
'+61 3 96852900', 1), (u'phone', u'+61 3 96960051\u200e', 1), (u'phone', u'+61 39690 2390', 1), (u'phone', u'+
613 9642 3272', 1), (u'phone', u'+613 9662 3888', 1), (u'phone', u'+61394177557', 1), (u'phone', u'+6139419300
0', 1), (u'phone', u'+61396399600', 1), (u'phone', u'+61396636263', 1), (u'phone', u'+61396865223', 1), (u'pho
ne', u'0393477922', 1), (u'phone', u'0394196118', 1), (u'phone', u'0394198600', 1), (u'phone', u'0396022228',
1), (u'phone', u'0396294111', 1), (u'phone', u'0396297405', 1), (u'phone', u'0398272608', 1), (u'phone', u'0\\
61396461117', 1), (u'phone', u'9670 4442', 1)]
```

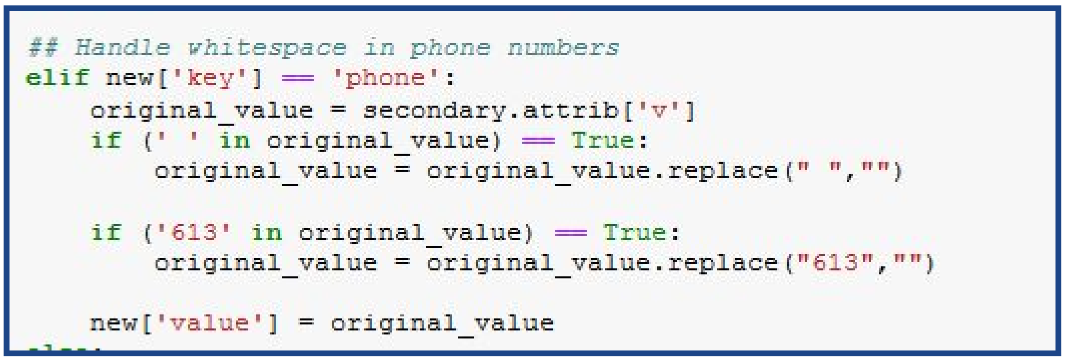
Some basic database querying revealed an inconsistency in phone numbers.



```
## Phone number inconsistency
cur.execute("SELECT key,value, COUNT(value) FROM ways_tags WHERE key ='phone' GROUP BY value ORDER BY
            COUNT(value) DESC")
all_rows = cur.fetchall()
print "Phone numbers ways_tags"
print all_rows, "\n"

cur.execute("SELECT key,value, COUNT(value) FROM nodes_tags WHERE key ='phone' GROUP BY value ORDER
            BY COUNT(value) DESC")
all_rows = cur.fetchall()
print "Phone numbers nodes_tags"
print all_rows, "\n"
```

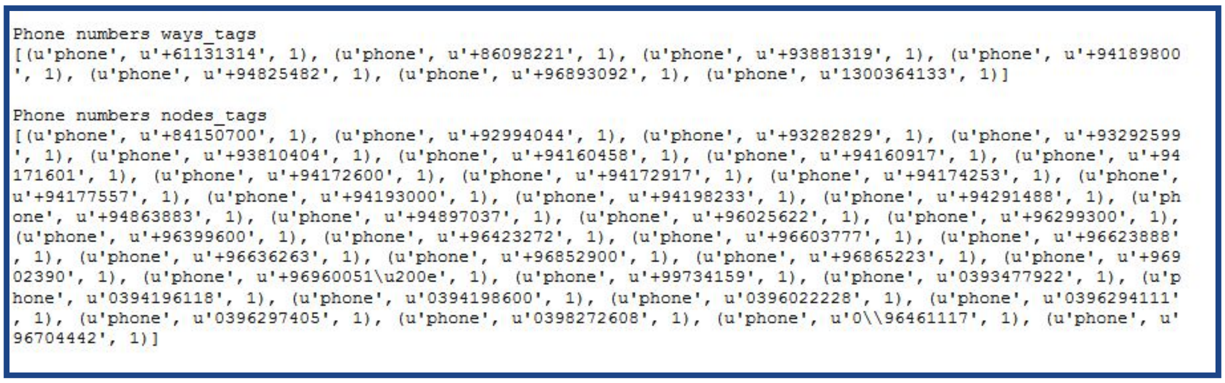
Some phone numbers contain one or more whitespaces within the numbers while some are joined as one full number. This can be a problem because first of all when just using programming to analyze the data such using regex characters only one part of the number can show up and also it is messy data to work with. Also another issue is some of the numbers contain the area code (613) while many don’t. So I’m going to strip any whitespaces and the area code from the numbers, so they’re all 8 digit numbers. It is better to keep them all consistent and in one manner.

- ***The code for removing whitespaces and area codes:***



```
## Handle whitespace in phone numbers
elif new['key'] == 'phone':
    original_value = secondary.attrib['v']
    if (' ' in original_value) == True:
        original_value = original_value.replace(" ","")

    if ('613' in original_value) == True:
        original_value = original_value.replace("613","")

    new['value'] = original_value
```

- ***New phone-numbers in the database:***



```
Phone numbers ways_tags
[(u'phone', u'+61131314', 1), (u'phone', u'+86098221', 1), (u'phone', u'+93881319', 1), (u'phone', u'+94189800
', 1), (u'phone', u'+94825482', 1), (u'phone', u'+96893092', 1), (u'phone', u'1300364133', 1)]

Phone numbers nodes_tags
[(u'phone', u'+84150700', 1), (u'phone', u'+92994044', 1), (u'phone', u'+93282829', 1), (u'phone', u'+93292599
', 1), (u'phone', u'+93810404', 1), (u'phone', u'+94160458', 1), (u'phone', u'+94160917', 1), (u'phone', u'+94
171601', 1), (u'phone', u'+94172600', 1), (u'phone', u'+94172917', 1), (u'phone', u'+94174253', 1), (u'phone',
u'+94177557', 1), (u'phone', u'+94193000', 1), (u'phone', u'+94198233', 1), (u'phone', u'+94291488', 1), (u'ph
one', u'+94863883', 1), (u'phone', u'+94897037', 1), (u'phone', u'+96025622', 1), (u'phone', u'+96299300', 1),
(u'phone', u'+96399600', 1), (u'phone', u'+96423272', 1), (u'phone', u'+96603777', 1), (u'phone', u'+96623888'
, 1), (u'phone', u'+96636263', 1), (u'phone', u'+96852900', 1), (u'phone', u'+96865223', 1), (u'phone', u'+969
02390', 1), (u'phone', u'+96960051\u200e', 1), (u'phone', u'+99734159', 1), (u'phone', u'0393477922', 1), (u'p
hone', u'0394196118', 1), (u'phone', u'0394198600', 1), (u'phone', u'0396022228', 1), (u'phone', u'0396294111'
, 1), (u'phone', u'0396297405', 1), (u'phone', u'0398272608', 1), (u'phone', u'0\\96461117', 1), (u'phone', u'
96704442', 1)]
```

Improving and analyzing data

Since phone numbers have an inconsistency in this data set perhaps we can use a phone book to improve the data. We can find a place in the map, see if it has an inconsistent number (e.g. white spaces, missing area code) and try to find that place in the phone book and then match the number for accuracy or make sure we have the correct number.

Benefits:

- Phone books contain consistent numbers in one format so using them to validate the map, can provide the map or data imported from the map with accurate consistent phone numbers
- Phone books also contain other information such as owner of a place and so we can add new "k" values(<tag k= "name_of_owner">) to the data

Problems:

- It would be difficult to find a phone book that contains all data and/or places in the map
- To iterate through every number and matching them with numbers in the phone book would require a lot of additional time and code, since instead of just exporting the data from the XML file to the csv file, we now have to check the numbers against the phone book before the data can exported to the csv file

Data Summary and Interesting Facts

WHERE K = 50

Number of unique users	***Number of nodes***	***Number of ways***
257	4687	815

Top ten contributing users	***File sizes***
Top ten contributing users [(u'Canley', 605), (u'Leon K', 561), (u'melb_guy', 514), (u'AlexOnTheBus', 338), (u'stevage', 230), (u'matthewsheffield', 193), (u'Mikideez', 176), (u'woowoowoo', 144), (u'Neil Penman', 143), (u'Pizza1016', 130)]	Melbourne-Map : 59,167KB Melbourne-Map-Sample : 1,182 KB Mydv : 661 KB Nodes.csv : 388 KB Nodes_tags.csv : 70 KB Ways: 49 KB Ways_nodes: 142 KB Ways_tags: 88 KB

Interesting Facts

Amenities:

Amenities in Melbourne from nodes tags	Number of different amenities from nodes tags
[(u'amenity', u'atm', 1), (u'amenity', u'bank', 2), (u'amenity', u'bar', 2), (u'amenity', u'bbq', 1), (u'amenity', u'bench', 8), (u'amenity', u'bicycle_parking', 7), (u'amenity', u'bureau_de_change', 1), (u'amenity', u'cafe', 22), (u'amenity', u'car_wash', 1), (u'amenity', u'college', 1)]	120

Amenities in Melbourne from ways tags	Number of different amenities from ways_tags
[(u'amenity', u'bank', 1), (u'amenity', u'cafe', 1), (u'amenity', u'casino', 1), (u'amenity', u'conference_centre', 1), (u'amenity', u'doctors', 1), (u'amenity', u'fuel', 1), (u'amenity', u'hospital', 1), (u'amenity', u'parking', 19), (u'amenity', u'place_of_worship', 1), (u'amenity', u'public_building', 2), (u'amenity', u'school', 2), (u'amenity', u'shelter', 2), (u'amenity', u'theatre', 1), (u'amenity', u'university', 1)]	35

The amenities are different in nodes_tags and ways_tags. Overall popular amenities are banks, ATM's, and bars.

Sources:

Sources in Melbourne from nodes tags	Number of different sources from nodes_tags
[(u'source', u'Bing', 2), (u'source', u'Collected via KeypadMapper', 2), (u'source', u'GPS', 1), (u'source', u'http://yarratrams.com.au/using-trams/service-changes/service-changes/2015/route-55-network-upgrade-work-saturday-23-to-friday-29-may/', 1), (u'source', u'https://www.ptv.vic.gov.au/live-travel-updates/article/route-55-temporary-tram-stop-closure-from-monday-13-february-2017-to-late-2019/', 1), (u'source', u'nearmap', 7), (u'source', u'survey', 9)]	23

Top 10 Sources in Melbourne from ways tags	Number of different sources from ways_tags
[(u'source', u'nearmap', 66), (u'source', u'Yahoo', 44), (u'source', u'Bing', 29), (u'source', u'survey', 11), (u'source', u'MMBW', 8), (u'source', u'default residential speed limit in Australia', 4), (u'source', u'ABS2011', 3), (u'source', u'GPS', 2), (u'source', u'bing,collins 1936', 2), (u'source', u'bing,knowledge', 2), (u'source', u'Direct', 1), (u'source', u'NearMap', 1)	182

Networks:

Networks in Melbourne from nodes tags	Number of different networks from nodes_tags
[(u'network', u'PTV - Metropolitan Trams', 32), (u'network', u'PTV - Metropolitan Buses', 12), (u'network', u'PTV - Metropolitan Trains', 3), (u'network', u'PTV', 2), (u'network', u'PTV - Regional Trains', 1)]	50

Networks in Melbourne from ways_tags	Number of different network from ways_tags
[(u'network', u'S', 15), (u'network', u'PTV - Metropolitan Trams', 8)]	23

Conclusion

Comparing with datasets we used in the classes this was a much clear and consistent dataset to clean. However, there were some minor inconsistencies we saw i.e. “addr:state” values were written in two different ways and same values for source keys were written in multiple ways.

Firstly, I used Python to get a rough look at the dataset e.g. get the different tag names, the keys in the tags and their values, get a count of each value and look for inconsistency. Then I created functions such as clean_tag() and shape_element() to clean the data and put it in a csv file.

After exporting from the csv files to database some basic queries revealed some errors such as the inconsistency in the source key (found same tags with different values and different counts within the ways and nodes parent tags. Also, another queries revealed interesting facts such as popular amenities, networks and top contributors to the dataset.

Overall this was a straightforward dataset to clean and the process and code can be iterated over and over to make any new adjustments to improve the shape of the data.