Shakir Alharthi – Mar-2016
Project 3- Wrangle OpenStreetMap Data

Introduction:

I started the project by one idea that dominated me and derailed me of the project for a while.
I asked myself: how can I view only the skeleton of a large XML file without reading its content and without opening it in an XML reader program.
So I searched about that and found some free software mentioned on this StackOverflow question:
http://stackoverflow.com/questions/308679/lightweight-xml-viewer-that-can-handole-large-files

However my concern was still valid: How can I see the skeleton only; without opening the tree of the XML and explore every node one by one on the elements tree on any XML file reader.

So, I decided that I can do it !

My answer to my question was implemented on this code on GitHub using python :
https://github.com/alshakir/xmlSkeleton_py

After all, that took me awhile to finish it; then I returned back to this course project with clear mind :)

So regarding this project; since I am from Saudi Arabia I chose the capital city : Riyadh.

I downloaded the file named: *riyadh_saudi-arabia.osm.bz2* that is 4.82 MB compressed and 71 MB uncompressed.

Once uncompressed I used my code 'xmlSkeleton_py' mentioned earlier to have a clue to the shape and skeleton of this XML file. This came compatible with the OSM wiki documentations.

Problems encountered:

Detailed code and explanation is found on Git-Hub here:
https://github.com/alshakir/DAND_mongo/blob/master/wrnagleOpenStreetMap-code_.ipynb

1- Viewing and browsing:

At the beginning, I used the notepad++ to read the file manually, and that was difficult because the file is large and it became slow for scrolling. At this moment, I realised why we need some libraries to deal with large data files.

2- Unicode naming and Arabic spelling conventions:

One of the problem I found is that since this is an Arabic city; the conversion of names from Arabic to English is not consistent,. For example if a street is named after a person the spelling of the name can be different depending on the person who converted it. To give a simple example, “King **Fahd** Street “ can be written as “King **Fahad** Street” or “King **Fahaad** Street”, because there is no rule on how to convert Arabic names to English spelling. So, I couldn't find a good plan to correct any Arabic name based on a solid scientific plan. So I opted to ignore the problem in English/Arabic names spelling. However, I did correct any naming abbreviations like St. to Street etc.

3- Abnormal zip code:

Starting to investigate any abnormal Zip-Code entry; I referred to Saudi post guidelines about the zip code on the following link: http://www.address.gov.sa/en/address-format/zip-code.

The convention is easy : five digits starting by the number '1'.
I found three abnormal entries. Tow of them only had other additions : 12783-8458 and 12393 4057. The third one was from 'Italy' and the city was Carrara. This meant that I need to investigate the user who input this and all of his/her other nodes

4- Problematic keys:

Using the code in the project exercises, I could reach to the fact that there was only one abnormal key; which was 'name 12/30' with the value 'unclassified'. Again this meant that I need to investigate the user who input this and all of his/her other nodes

5- Investigating the users who input wrong zip-code or problematic keys.

I developed a function to print the values of some suspected tags. The function name is ***'printValuesOfTheKey(filename, keyForValue)'***
I applied with some tags keys that seemed abnormal like *'fixeme'*, *'FIXME'*, or Unicode keys like *u'\u0627\u0644\u0644\u0648\u064a\u0645\u064a'*

Also to capture the element that contains abnormal or problematic keys I developed the function named: ***'extractParentElementsByKey(filename, key)'***
I used it on the problematic key 'name 12/30' to find that the element was added by 'uid: 111462, user: Bot45715'

6- Browsing large list on IPython results in crashing of the browser.

When I wanted to check all the input elements of the user Bot45715 (uid: 111462). I developed the function ***'findUserNodes(filename, userID)'*** to scroll through the nodes and find any suspected input.

During the execution of this method I found that if the user had large number of input nodes then we can't view it on the IPython notebook because it will create a large DOM nodes and that will clog the browser or crash it. So I developed another method that will read only a chunk of the user nodes based on the some parameters. The function named :***'giveMeChunkOfTheList(theList, theChunk, thePart = 0)***'

7- Unexpected node types:

Elements selected to migrate to Mongodb are of type 'node' and 'way' using the element names (node and way). however I found that if there is a tag element that contains a 'k' attribute named : type; then its 'v' value will override the value of the dictionary key names “type”. This was managed through the code. So if any tag contained a 'k' attribute named 'type' it will be added to the dictionary as 'type-userdefined'

8- Aside problem:

Unicode characters view on the cmd.exe shell (mongo shell) windows 10 couldn't show the Arabic letters.

After searching for many hours about it especially on: http://superuser.com/questions/157225/even-on-windows-7-can-you-do-a-dir-and-be-able-to-see-filenames-that-has-unic

I reached to the conclusion that :'The reason is that CMD cannot display Unicode. It can, however, display DBCS (Double-Byte Character Set)'.

The indirect solution to this was to pipe the output to a text file and view it through the notepad, as suggested in the link above. I wish a day will come where the Unicode char-set will be the default for every computer system on the planet.

Overview of the data and Statistics.

The OSM file size was 4.82 MB compressed and 71 MB uncompressed.

Total items = 390459
Node type count = 317700
Way type count = 72759

Nodes percentage = 81.37 %
ways percentage = 18.63 %

Most frequent places entered as amenities.

Using the following query:

```
amenities = list(db.osm2.aggregate([{"$match":{"amenity":{"$exists":1}}},{"$group":{"_id":
"$amenity",\
        "count":{"$sum": 1}}}, {"$sort": {"count": -1}}]))
```

Total places : 47

Sample of the result:

'place_of_worship': 978
'parking': 726
'school': 219
'post_box': 2
'bus_station': 2
'courthouse': 1
Notice that Riyadh is a large city with a population of more than 7 million. So these data are not accurate apparently since there are hundred of bus stations and courthouses and post offices.

Total users

Query:

users = list(db.osm2.aggregate([{"$group":{"_id": "$created.user", "count":{"$sum": 1}}}, {"$sort": {"count": -1}}]))

Total users = 282
The top 5 users are:
'Seandebasti','count': 174192
'Rub21', 'count': 44976
'bauma', 'count': 30582
'Cicerone', 'count': 27862
'hsarslan', 'count': 17825

Notice that the user **Seandebasti** has input 44% of the total nodes. And most users input only a few nodes. That is why the histogram graph of user input was skewed to right. [Graph on the code page]

Religion percentage

For curiosity, and since most frequent node amenity is 'place of worship', I tried to see if there are different religion places for worship??

```
list(db.osm2.aggregate([{"$match":{"amenity":{"$exists":1},"amenity":"place_of_worship"}},\
                {"$group":{"_id":"$religion", "count":{"$sum":1}}}]))
```

result was:
'muslim', 'count': 969
None, 'count': 9

However in reality there is no PUBLIC place of worship in Riyadh other than Mosques.

Other ideas

I think that there should be a template for the common places, for the nodes to choose from. like school, restaurant, cafe, park, so each template has its own mandatory tags specific to it, that must be filled in, with the possibility to add other optional tags. The node shouldn't be considered valid unless all the essential tags are filled. This way we prevent the strange and abnormal tags by some users. Although we can't guarantee that the user will abide by the template; however, we guide the user toward a better clean input.

Another idea to improve the authenticity of data is by asking for the users help (even non-registered users) about the correctness of the item when they view the details. To implement this there should be a toggle button that mark the node as correct/incorrect, when the user click on a node to view details, and every element correctness is measured based on the ratio of incorrect to correct scores [I will call it 'negativity']. e.g.:
1- Node A { correct: 120, incorrect: 88} => with 73% of negativity
2- Node B { correct: 10, incorrect: 98} => with 980% of negativity
2- Node C { correct: 1050, incorrect: 1018} => with 97% of negativity

Here node A is the most trusted since it got low negativity.

They key factor for success is that we allow any non-registered user to toggle the button to inform us about wrong naming or attributes. Users tend to like interactive opinion buttons, but they are thwarted by log-in or identity screens.