Bangalore is where I live now and I am curious to learn more about the city through OSM. I would also like to understand the opportunities to contribute to the information about Bangalore on OSM.

1. Map Area: Bangalore, India

Downloaded Data:
Overpass API Query:

```
<osm-script timeout="900" element-limit="1073741824">
  <bbox-query s="12.83" w="77.56" n="12.98" e="77.7"/>
  <print/>
</osm-script>
```

The data was loaded into Python and simple queries were run using the Element Tree module.

File Size: 51.6 MB

Queries:

```
import xml.etree.ElementTree as ET
name_file = "blore_select"
tree = ET.parse(name_file)
```

2. General Queries – Exploring the XML file (SAVED IN OSMCHECKS.PY FILE)

Queries:

```
tags_list = []
for child in root:
    if child.tag not in tags_list:
        tags_list.append(child.tag)

print tags_list

count_tag = 0
for i in tree.getiterator('tag'):
  count_tag = count_tag + 1
count_node= 0

for i in tree.getiterator('node'):
  count_node = count_node + 1
print count_tag
print count_node
```

Observation 1: There are no ways or relations tags in the data

It was found that the data contains no ways or relations tags. More than 95% of the elements are 'node' tags, and a majority of them contain only latitude and longitude information.

Root tag:
'osm'

Child tags:
['note', 'meta', 'node']

Total number of rows: 8,69,934
Rows with element tag 'node' = 8,32,892
Rows with element tag 'tag' = 37,039
Rows with element tag 'osm' = 1
Rows with element tag 'meta' = 1
Rows with element tag 'note' = 1

There is no way to fix this. Only the user entering the information can address this.

Observation2: There is no ‘user’ or ‘uid’ information in the nodes tags.

The following queries returned none. This means that the user has not entered their own information and we cannot do anything about it.

Queries

```
for i in tree.getiterator('node'):
   c1 = i.get('uid')
   if c1:
      print c1
```

Queries

```
for i in tree.getiterator('node'):
   c1 = i.get('user')
   if c1:
      print c1
```

Observation3: The tag element only contain [‘k’,’v’] attributes.
Since nodes do not contain much information, we will now look into the “tag” elements.

Queries

```
attribs = []
for i in tree.getiterator('tag'):
   if i.keys() not in attribs:
      attribs.append(i.keys())
print attribs

typedict_k = []
for i in tree.getiterator('tag'):
   c1 = i.get('k')
   type_i = type(c1)
   if type_i not in typedict_k:
      typedict_k.append(type_i)

typedict_v = []
for i in tree.getiterator('tag'):
   c2 = i.get('v')
   type_i = type(c2)
   if type_i not in typedict_v:
      typedict_v.append(type_i)

print typedict_k
print typedict_v
```

On printing the attribute names of all the tag elements, we find that the tag element only contain [‘k’,’v’] attributes.

Checking the types of the tag attributes:
k values: [<type 'str'>]
v values: [<type 'str'>, <type 'unicode'>]

We have to remember to use Unicode supporting csv while exporting data to csv file.

Observation 4: Checking for data types and primary key in node element.

Checking the data types of attributes of the ‘node’ elements:
id: [<type 'str'>]
lat: [<type 'str'>]
lon:[<type 'str'>]

Since id is the primary key, we also need to check that it is not Null for any value. It was found that id is present for all node element tags and there are no Null values.

3. Finding problems in the data for tag attributes 'k' and 'v' values, and correcting the data. (SAVED IN OSM2Output.PY FILE)

Counting initial number of total rows: 869934

Queries:

```
for i in root.iter():
   rows_initial = rows_initial + 1
print "Total number of initial rows"
print rows_initial
```

3.1 Checking and correcting for postal codes.

44 postal codes were found in non-standard format by using a match with the regular expression: ('^(5)(6)\d{4}$') This expression is based on the information that all Bangalore codes must start with the digits "56" and be six characters long. The last four characters must be digits.

```
initial_wrong_codes = []
codes_wrong_after_correction = []
count = 0
regex = re.compile('^(5)(6)\d{4}$')

for node in root.findall('node'):
   for tag in node.iter('tag'):
      k1 = tag.get("k")
      if k1 == "addr:postcode":
         v1 = tag.get("v")
         m1 = regex.match(v1)
         if not m1:
            initial_wrong_codes.append(v1)
            if len(v1) <> 6:
               v1 = v1.replace(" ","")
               v1 = v1.replace(",","")
               v1 = v1.replace("-","")
               v1 = v1.replace("'","")
               tag.set("v",v1)
               m2 = regex.match(v1)
               if not m2:
                  codes_wrong_after_correction.append(v1)
                  node.remove(tag)
                  count = count +1
            elif len(v1) == 6:
               node.remove(tag)
               count = count +1
               codes_wrong_after_correction.append(v1)


print "wrong codes list:"
print initial_wrong_codes
print "codes that cant be corrected"
print codes_wrong_after_correction
print "number of codes dropped"
```

After identifying the 44, special characters and spaces were replaced with "" and then we were left with 20 nonstandard postal codes as below: ['5600041', '5600011', '570008', '570008', '570008', '570008', '380068', '500006', '530078', '571438', '570008', 'Bengaluru', '5600109', '530103', '5600037', '380068', '583105', '5560034', '650027', '560001ph'] Since there is no way of correcting these pin codes, we will drop these rows from the data.

```
regex1 = re.compile('(8)|(08)')
countm1 = 0
m1list = []

regex2 = re.compile('(9)|(09)')
countm2 = 0
m2list = []

regex3 = re.compile('(7)|(07)|(1800)|(3)|(4)')
countm3 = 0
m3list = []

for node in root.findall('node'):
   for tag in node.iter('tag'):
      k1 = tag.get("k")
      if k1 == "phone":
         v1 = tag.get("v")
         v1 = v1.replace(" ","")
         v1 = v1.replace("  ","")
         v1 = v1.replace("-","")
         v1 = v1.replace('+',"")
         v1 = v1.replace("(91)", "91")
         v1 = v1.replace("0091", "91")
         v1 = v1.replace("O", "0")
         v1 = v1.replace("'", "")
         v1 = v1.replace('01800', "1800")
         v1 = v1.replace('0099', "99")
         v1 = v1.replace('9180', "80")
         v1 = v1.replace('91080', "80")
         tag.set("v",v1)


## Now applying regex:

         m1 = regex1.match(v1)
         m2 = regex2.match(v1)
         m3 = regex3.match(v1)

         if m1:
            if len(v1) <= 9:
               countm1 = countm1 + 1
               m1list.append(v1)
               node.remove(tag)
            elif len(v1) >= 12:
               if v1.find(',') == -1:
                  v2 = v1[:11]+", "+v1[11:]
                  if len(v2) < 20:
                     countm1 = countm1 + 1
                     m1list.append(v1)
                     node.remove(tag)
         elif m2:
            if v1[:2] == "91" and len(v1) <> 12:
               if v1[:6] <> "911800":
                  if len(v1) <> 25 or len(v1) <> 38 or len(v1) <> 51:
                     v2 = v1[:12] + "," + v1[12:]
                     if v2[13:15] == '91':
                        if len(v2[13:]) == 12:
                           v1 = v1.replace(v1,v2)
                           tag.set("v",v1)
                        else:
                           countm2 = countm2 + 1
                           m2list.append(v1)
                           node.remove(tag)
                     elif v2[13:15] == '97':
                        if len(v2[13:]) == 10:
                           v1 = v1.replace(v1,v2)
                           tag.set("v",v1)
                        else:
                           countm2 = countm2 + 1
                           m2list.append(v1)
                           node.remove(tag)
                     elif v2[13:16] == '080' or v2[13:15] == '80' or  v2[13:17] == ',080':
                        v1 = v1.replace(v1,v2)
                        tag.set("v",v1)
                     else:
                        countm2 = countm2 + 1
                        m2list.append(v1)
                        node.remove(tag)
         elif m3:
            if len(v1) == 8:
               v2 = '80'+ v1
               v1 = v1.replace(v1,v2)
               tag.set("v",v1)
         else:
            countm3 = countm3 + 1
            m3list.append(v1)
            node.remove(tag)

print "Phone numbers with starting digit 8,9, and others that can't be corrected, hence
dropped"
print countm1
print m1list
print ""
print countm2
print m2list
print ""
```

3.2 Checking and correcting for <u>phone numbers</u>:

Landline numbers in Bangalore must start with 80, or 080. Similarly, mobile numbers should start with 91 for India code. Length of mobile numbers should be 10, and of landline 8. Using these conditions, the incorrect values were identified.
There were cases where two numbers were added without any commas, this has also been corrected for.
There were many problems found with phone numbers and that has been corrected using the code below. 21 phone numbers could not be corrected, and have been removed completely. The removed list after all the corrections is:

9 starting with 80/080
['809590160160', '080380000', '804302501', '0806502306697', '806656000', '080880533338', '08:00to21:00', '802225874', '80425655']

9 starting with 9
['9143336000', '916604797', u'91\xa08025536090', '9164425865', '9108129532', '9148100961', '91234156789', '9164212461', '9148971958']

3 others
['123456789', '123456789', '6747589579869']

4.3 Similarly checking for duplication in amenity names we found a couple categories repeating. Fixing it as with code below:

Queries:

```
for for node in root.findall('node'):
    for tag in node.iter('tag'):
        k1 = tag.get("k")
        if k1 == "amenity":
            v1 = tag.get("v")
            if v1 == 'ice cream':
                v1 = v1.replace(v1, "ice_cream")
                tag.set("v",v1)
            elif v1 == "bar" or v1 == "pub":
                v1 = v1.replace(v1, "bar;pub;restaurant")
                v1 = v1.replace(v1, "bar;pub;restaurant")
                tag.set("v",v1)
```

In total 41 rows were dropped. Checking the number of lines in the resulting file:

Queries:

```
for i in root.iter():
    rows_final = rows_final + 1
print "Total number of final rows"
print rows_final
```

869893

This is correct, as in total out of 869934, we dropped a total of 41 rows.

4.4 Exporting data to csv

Codefile: Output2CSV.py

Data was exported to two csv files.
OSM_nodes.csv has fields ID, Lat, Lon and has 832892 rows; file size: 28.3 MB
OSM_nodes_tags.csv has fields ID, k,v, Type fields, and has 36998 rows; file size: 1.7 MB

In the original file, there were 37039 rows. 41 were dropped, hence tag rows decreased to 36,998.

The numbers of rows match the number of rows in xml. The data was imported correctly.

```
import xml.etree.ElementTree as ET
import unicodecsv as csv

### Writing the file into csv format
name_file = "Output.xml"
tree = ET.parse(name_file)
root = tree.getroot()


OSM_nodes = open('OSM_nodes.csv', 'w')

csvwriter = csv.writer(OSM_nodes)
osm_nodes_header = []

for i in tree.getiterator('node'):
    attribute_nodes = i.keys()
for i in attribute_nodes:
    osm_nodes_header.append(i)

csvwriter.writerow(osm_nodes_header)
count = 0
for row in root.findall("node"):
    osm_row = []
    lat = row.get("lat")
    lon = row.get("lon")
    id = row.get('id')
    osm_row = [lat, lon, id]
    csvwriter.writerow(osm_row)
    count = count+1

print "Total rows in nodes file:"
print count
print ""

OSM_nodes_tags = open('OSM_nodes_tags.csv', 'w')
csvwriter2 = csv.writer(OSM_nodes_tags)
osm_nodes_tags_header = ["id", "v_type"]

for i in tree.getiterator('tag'):
    attribute_nodes = i.keys()
for i in attribute_nodes:
    osm_nodes_tags_header.append(i)

csvwriter2.writerow(osm_nodes_tags_header)
count2 = 0
for row in root.findall("node"):
    id = row.get('id')
    for line in row.findall("tag"):
        osm_row = []
        k = line.get("k")
        v = line.get("v")

        v_type = type(v)
        id1 = id
        osm_row = [id1,v_type, k,v]
        csvwriter2.writerow(osm_row)
        count2 = count2+1

print "Total rows in tags file"
print count2

print count+count2
```

3.2 Creating tables into SQLite3:

<u>**Queries**</u>

```
CREATE TABLE nodes (
            lat,
            lon,
            id PRIMARY KEY NOT NULL
        );


        CREATE TABLE nodes_tags (
            id,
            type,
            key,
            value,
            FOREIGN KEY (id) REFERENCES nodes(id)
        );
```

3.2 Importing data from CSV into SQL tables

<u>**Queries**</u>

```
.mode csv
.import osm_nodes.csv nodes
.import osm_nodes_tags.csv nodes_tags
```

checking if all the rows have been imported:

```
sqlite> select count(*) from nodes;
832893
sqlite> select count(*) from nodes_tags;
36998
```

We can also check for some corrections that we made, and see if the correct values have been imported into SQL.

If we loaded the data correctly, these queries should return null.
1. length of postal codes > 6:
select * from nodes_tags where key = "addr:postcode" and length(value)>6;
2. pub and bar values in amenity:
select * from nodes_tags where key = "amenity" and value = "bar";
select * from nodes_tags where key = "amenity" and value = "pub";

4. Insights from sql data:

4.1 User data:

User called JOSM has submitted the highest number of queries.

select value, count(*) from nodes_tags where key = "created_by" group by value;

JOSM,413

```
"Potlatch 0.10c",2
"Potlatch 0.10d",1
"Potlatch 0.10e",8
"Potlatch 0.10f",14
"Potlatch 0.7b",2
"Potlatch 0.9",2
"Potlatch 0.9a",7
"Potlatch 0.9b",1
"Vespucci 0.6.5",1
cap4access,1
"iLOE 1.9",1
```

4.2 Top 20 keys

```
select key, count(*) from nodes_tags group by key order by count(*) desc limit 20;
name,8041
amenity,4206
addr:street,2390
shop,2166
natural,1846
addr:postcode,1505
addr:city,1442
addr:housenumber,1230
highway,1131
phone,1111
name:kn,1055
cuisine,748
website,702
opening_hours,684
level,598
operator,561
office,535
created_by,453
power,436
barrier,343
```

4.3 Top 10 street addresses

```
select value,count(*) from nodes_tags where key = "addr:street" group by value order by count(*) desc limit 10;

"Chinmaya Mission Hospital Road",67
"100 Feet Road(S K Karim Khan Road)",61
"Bannerghatta Road",61
"Magrath Road",48
"Hosur Road",35
"Outer Ring Road",34
"100 Feet Road( S K Karim Khan Road)",32
"Sarjapur Road",28
"12th Main Road",21
"80 Feet Road(Sir C.V. Raman Hospital Road)",21
```

4.4 Top 10 postal codes:

```
sqlite> select value,count(*) from nodes_tags where key = "addr:postcode" group by value order by count(*) desc limit 10;

560103,127
560076,116
560037,102
560102,101
560004,85
560095,83
560025,79
560034,73
560100,67
560068,64
```

4.5 Top 10 cuisines

```
indian,171
regional,148
vegetarian,52
chinese,43
pizza,43
ice_cream,31
coffee_shop,26
burger,25
italian,23
international,21
```

4.6 Most marked places with names

```
select value,count(*) from nodes_tags where key = "name" group by value order by count(*) desc limit 10;

"Cafe Coffee Day",43
"State Bank of India",33
"Canara Bank",29
"ICICI Bank",26
"HDFC Bank",22
"Apollo Pharmacy",21
Palm,20
"Street Lamp",20
"Axis Bank",18
"Corporation Bank",18
```

4.7 Top 10 amenities

```
select value,count(*) from nodes_tags where key = "amenity" group by value order by count(*) desc limit 10;

restaurant,922
atm,393
bank,365
place_of_worship,291
pharmacy,235
fast_food,231
hospital,209
bench,168
cafe,163
```

```
school,125
```

4.8 Top 10 shops

```
select value,count(*) from nodes_tags where key = "shop" group by value order by count(*) desc limit 10;
```

```
clothes,290
supermarket,191
bakery,140
beauty,112
convenience,99
sports,93
electronics,71
jewelry,61
greengrocer,54
shoes,53
```

4.9 Top 10 street features

```
sqlite> select value,count(*) from nodes_tags where key = "highway" group by value order by count(*) desc limit 10;
```

```
bus_stop,578
traffic_signals,255
street_lamp,234
turning_circle,26
crossing,20
mini_roundabout,7
motorway_junction,7
services,2
junction,1
residential,1
```

4.10 Top 10 power related landmarks

```
select value,count(*) from nodes_tags where key = "power" group by value order by count(*) desc limit 10;
```

```
tower,373
pole,30
transformer,24
sub_station,6
generator,2
substation,1
```

Conclusion

These results show that the places that have been covered most are Bellandur and Indirangar. Both these places have lots of shopping areas and also are premium residential areas. It was interesting to see that restaurants are highest occurring amenities, and café coffee day – a very popular coffee shop in Bangalore, is one of the most marked places.

Possible Improvement

There are no "ways" and "relations" tags. In general, the data for Bangalore on OSM is very limited.

Some improvements to the data and data analysis process that could be useful:

1) We dropped several pin codes because of inconsistencies. But since we know the address/area/geographical coordinates of the places, we can actually easily google up the pin codes and add or correct them. It would be really interesting to do this programmatically.
2) A lot of places like ice cream shops and coffee shops have not been tagged as amenities. But from their names like "Naturals Ice Cream Store" or "Kwality ice cream parlour" we can easily make out that they are ice creams shops. We could improve the quality of the data by running simple queries on common words like 'café', 'ice cream' in the tags list and add 'amenity' tag if missing.
3) Number of restaurants, cafes, banks, etc. seem like a good indication of the development status of an area. The most developed areas in the city have the highest number of amenities. It would be interesting to do an analysis where various pin codes of the city are compared for various amenities. This could give an insight into the most and the least commercially developed areas. This data could be useful for advertisers/real estate brokers/people looking to buy a house.

Anticipated problems in implementing the improvements:

1) Upon adding pin codes we may realize that previously entered pin codes in the same area are incorrect and also need to be corrected for.
2) There is a chance that some areas have not been covered for at all in open street maps, and have nothing to do with the development index of that particular place. We saw that at least for Bangalore the most commercially active places have the most data (especially amenities). But it may not always be true. It would be great if Open Street Maps could provide an indication of how 'mature' the data is for a particular city, or maybe we can estimate that by using indicators such as - timeline of contribution, number of contributors, etc.. If regular updates have been made to the data for 3-4 years, or if say more than ten people have contributed to the data - maybe it can be considered sufficient for this analysis. These numbers need to be calculated based on a study of the data available for cities across the world.