

OpenStreetDataMap Report

database: MongoDB

location: Gurugram, Haryana, India

Name: Mayukh Sarkar

Email: mayukh2012@hotmail.com

Table of Contents

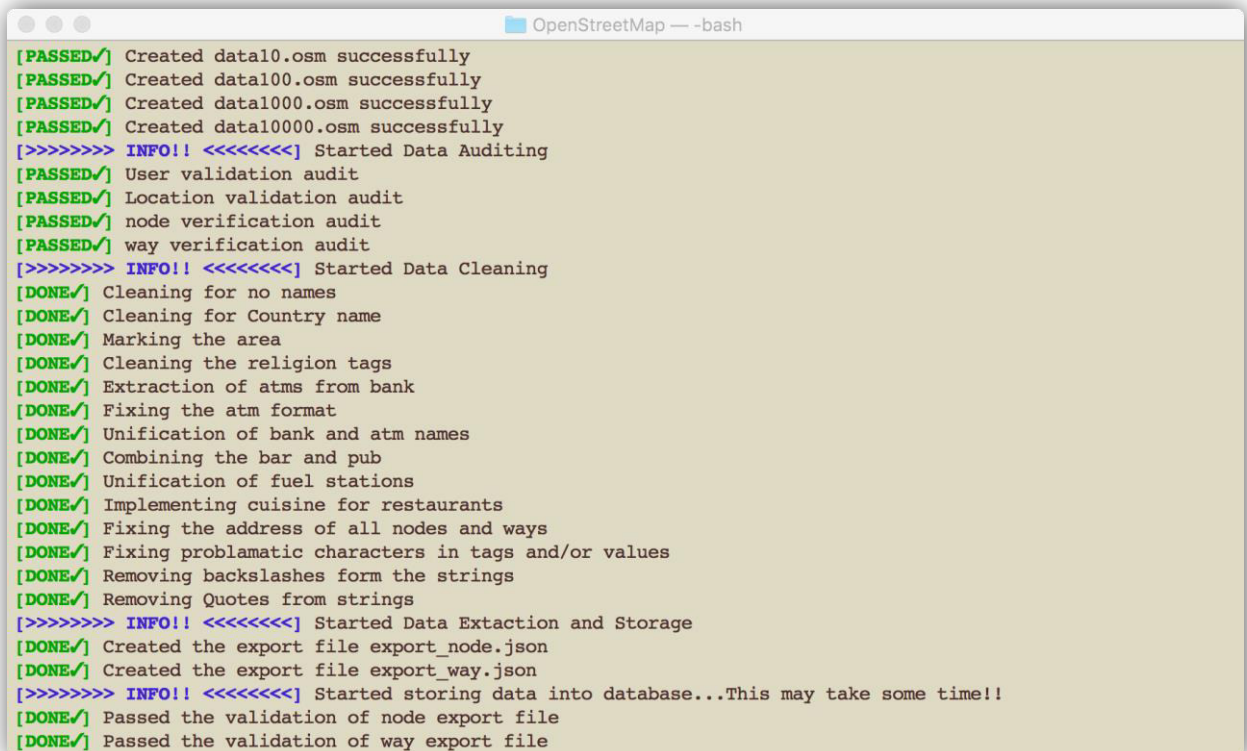
Introduction.....	1
Problems and Cleaning	2
▪ Amenities with no names	3
▪ Bad tag & value for country	3
▪ Place of worships without religion	4
▪ Errors in ATMs and banks	5
▪ Bar and Pub ambiguity	6
▪ Problematic characters	7
MongoDB Statistics	7
▪ Basic Statistics on Database	7
▪ Additional Statistics on Database.....	8
Conclusion.....	11

Introduction

The location that is used in this analysis is a popular city in North India in the state of Haryana named Gurugram previously known as Gurgaon. As a major industrial hub, this place is a potential candidate as one of the most important driving force of North Indian economy and a place of lot of Data Scientists.

The OSM file of the location was not readily available and the data is requested specially. The required links are provided in the GitHub project repo. The source code is relatively difficult to comprehend and hence the readme should properly be followed. A more detailed doc would be released soon.

The cleaning process is being quite complex and relative to the location but is kept as simple as possible. A conventional output of an ideal execution would look like following

A terminal window titled 'OpenStreetMap — -bash' displays a series of log messages. The messages are color-coded: green for success ('[PASSED✓]' or '[DONE✓]'), blue for information ('[>>>>>> INFO!! <<<<<<<]'), and purple for warnings or errors. The logs show the successful creation of four OSM data files (data10.osm, data100.osm, data1000.osm, data10000.osm), followed by a data auditing phase (user, location, node, and way verification) and a data cleaning phase (removing names, country names, marking areas, cleaning religion tags, extracting and fixing ATM data, unifying bank/pub names, combining bar/pub, unifying fuel stations, implementing cuisine for restaurants, fixing addresses, removing problematic characters, removing backslashes and quotes). The process concludes with data export to JSON files and storage in a database, with final validation checks passed.

```
[PASSED✓] Created data10.osm successfully
[PASSED✓] Created data100.osm successfully
[PASSED✓] Created data1000.osm successfully
[PASSED✓] Created data10000.osm successfully
[>>>>>> INFO!! <<<<<<<] Started Data Auditing
[PASSED✓] User validation audit
[PASSED✓] Location validation audit
[PASSED✓] node verification audit
[PASSED✓] way verification audit
[>>>>>> INFO!! <<<<<<<] Started Data Cleaning
[DONE✓] Cleaning for no names
[DONE✓] Cleaning for Country name
[DONE✓] Marking the area
[DONE✓] Cleaning the religion tags
[DONE✓] Extraction of atms from bank
[DONE✓] Fixing the atm format
[DONE✓] Unification of bank and atm names
[DONE✓] Combining the bar and pub
[DONE✓] Unification of fuel stations
[DONE✓] Implementing cuisine for restaurants
[DONE✓] Fixing the address of all nodes and ways
[DONE✓] Fixing problematic characters in tags and/or values
[DONE✓] Removing backslashes form the strings
[DONE✓] Removing Quotes from strings
[>>>>>> INFO!! <<<<<<<] Started Data Extaction and Storage
[DONE✓] Created the export file export_node.json
[DONE✓] Created the export file export_way.json
[>>>>>> INFO!! <<<<<<<] Started storing data into database...This may take some time!!
[DONE✓] Passed the validation of node export file
[DONE✓] Passed the validation of way export file
```

As one can see that there are multiple steps of cleaning and the complete data manipulation process includes

- A. Data Framing** - Scaling the raw data in different factors to facilitate testing different data cleaning operations.
- B. Data Auditing** – This is used to remove any dirty data. There is a difference between dirty data and unformatted data. Data auditing is like logical error in a program. It is syntactically not incorrect but it is logically incorrect. This can be verified from the documentation. In the data auditing section, we are verifying if the data is following the documentation.
- C. Data Cleaning** – This is cleaning the data if for any unformatted node and way in the openstreetmap database. The detailed analysis is given in the following section.

Problems and Cleaning

Multiple data cleaning operations are performed. Let's look at them gradually.

- **Amenities with no names**

There are multiple nodes and ways which has no tag **name**. This is not helpful specially if the node represents an **amenity**. For example, if there is a restaurant but there is no name associated for the restaurant, chances are that the restaurant is not present or may be it is not a good one. In this cleaning we are removing all the amenities with no names. Amenities of **atm** is although neglected because names of the atms are distributed in tags **name** & **operator** both. Below are some example values with no names.

The function is being called from here

```
searchMatchTag = ['amenity', 'highway', ('landuse', 'commercial')]
data_node = self.node_data
data_way = self.way_data
for tag in searchMatchTag:
    data_node = clean_nodes_no_names(tag, data_node)
    data_way = clean_nodes_no_names(tag, data_way)
```

As we can see that function is called with tag values **amenity, highway & commercial landuse**.

This means names in these tags are important and if there is no name, that node is removed with an exception of the following nodes

```
<node changeset="27016634" id="248852583" lat="28.5336472" lon="77.152001"
timestamp="2014-11-25T08:58:34Z" uid="1830192" user="Warin61" version="5">
  <tag k="atm" v="yes" />
  <tag k="name" v="ICICI, SBI, Citibank," />
  <tag k="amenity" v="bank" />
</node>

<node changeset="8696687" id="1358147152" lat="28.4516536" lon="77.0749759"
timestamp="2011-07-11T19:15:32Z" uid="451671" user="Edolis" version="1">
  <tag k="amenity" v="atm" />
  <tag k="operator" v="HDFC Bank" />
</node>
```

As we can see that both of the above mentioned nodes have **amenity** as **atm**. The first node has name in it but the second node doesn't. But we can still predict the name from the **operator**. Hence amenities with atm is not handled.

- **Bad tag & value for country**

The country name related cleaning is performed in two functions, namely,

```

self.node_data = expand_country_name('addr:country', 'India', self.node_data)
self.way_data = expand_country_name('addr:country', 'India', self.way_data)
self.node_data = fixAddress(self.node_data)
self.way_data = fixAddress(self.way_data)

```

This converts the following node like this,

```

<node changeset="6914006" id="527994799" lat="28.430407" lon="77.0582574"
timestamp="2011-01-09T11:46:40Z" uid="17429" user="thevikas" version="3">
  <tag k="addr:city" v="Gurgaon" />
  <tag k="addr:street" v="Sector 46" />
  <tag k="addr:country" v="IN" />
  <tag k="addr:housenumber" v="1537" />
</node>

```



```

<node changeset="6914006" id="527994799" lat="28.430407" lon="77.0582574"
timestamp="2011-01-09T11:46:40Z" uid="17429" user="thevikas" version="3">
  <tag k="city" v="Gurgaon" />
  <tag k="street" v="Sector 46" />
  <tag k="country" v="India" />
  <tag k="housenumber" v="1537" />
</node>

```

Just observe how the country name is changed. Also this shows how **addr** is removed from each tags. That what **fixAddress** function is doing.

- Place of worships without religion

This cleaning was performed for the **amenity** of **place_of_worship**. All the nodes with place of worship should have a tag called **religion**. The idea of secularism is good but not very useful for amenity of palce_of_worship because each place of worship should be associated with its respective religion. Hence this cleaning operation is performed for the

- I. *Removing the nodes having amenity of pace_of_worship but no religion for that place of worship.*
- II. *If the religion is not present but if the name of the place of worship is present, then we also tried to predict the religion form the name of the place of worship.*

The following example shows how it actually cleans the data.

```

<node changeset="34445638" id="266441574" lat="28.5057002" lon="77.1756767"
timestamp="2015-10-05T12:44:42Z" uid="290680" user="wheelmap_visitor" version="2">
  <tag k="amenity" v="place_of_worship" />
  <tag k="created_by" v="Potlatch 0.9a" />
  <tag k="wheelchair" v="limited" />
</node>

```



Removed because no *religion* or *name* to predict the religion

```
<node changeset="3757536" id="484649690" lat="28.4248732" lon="77.1012155"
timestamp="2010-01-31T10:12:12Z" uid="110263" user="werner2101" version="2">
  <tag k="name" v="Sanatan Dharm Mandir" />
  <tag k="is_in" v="Sector - 56" />
  <tag k="amenity" v="place_of_worship" />
  <tag k="religion" v="hindu" />
</node>
```



Kept without any change because *religion* is present

```
<node changeset="34587866" id="3783155937" lat="28.5624309" lon="77.0533592"
timestamp="2015-10-12T11:23:23Z" uid="104344" user="Shubhangi" version="1">
  <tag k="name" v="Hanuman Temple" />
  <tag k="amenity" v="place_of_worship" />
</node>
```



Not removed because even though *religion* is not present but *name* is present and from the name, religion was predicted

```
<node changeset="34587866" id="3783155937" lat="28.5624309" lon="77.0533592"
timestamp="2015-10-12T11:23:23Z" uid="104344" user="Shubhangi" version="1">
  <tag k="name" v="Hanuman Temple" />
  <tag k="amenity" v="place_of_worship" />
  <tag k="religion" v="hindu" />
</node>
```

This prediction was also performed for *amenity* of *restaurants*. Some restaurants have no cuisine mentioned hence cuisine of these restaurants is predicted from their names. Restaurants with no cuisine are not removed but rather kept with cuisine being **regional**. This is because in India, if a restaurant is small, they mostly offer regional or local food cuisine and restaurant with a specific cuisine is considered lavish.

▪ Errors in ATMs and banks

The problem with the amenity *atm* is that there are some entries with amenity of atm directly but there are some other entries with amenity as *bank* with a tag called *atm* to be **true**. Like the following.

```
<node changeset="2318444" id="477905082" lat="28.3936446" lon="77.0468278"
timestamp="2009-08-30T20:20:12Z" uid="26562" user="Nishant Sharma" version="1">
  <tag k="atm" v="yes" />
  <tag k="name" v="Union Bank of India" />
  <tag k="amenity" v="bank" />
```

</node>

Now of course those entries of banks should be considered **both as bank and atm** separately. Hence cleaning was performed to create two nodes from a single node like the following

```
<node changeset="2318444" id="477905082" lat="28.3936446" lon="77.0468278"
timestamp="2009-08-30T20:20:12Z" uid="26562" user="Nishant Sharma" version="1">
  <tag k="name" v="Union Bank of India" />
  <tag k="amenity" v="bank" />
</node>
```

```
<node changeset="2318444" id="477905082" lat="28.3936446" lon="77.0468278"
timestamp="2009-08-30T20:20:12Z" uid="26562" user="Nishant Sharma" version="1">
  <tag k="name" v="Union Bank of India" />
  <tag k="amenity" v="atm" />
</node>
```

There were more problems in atm or bank **names**.

```
<node id="4047437970" lat="28.5332679" lon="77.1519572" version="1"
timestamp="2016-03-08T07:52:33Z" changeset="37680885" uid="3691301"
user="Aksheyta">
  <tag k="name" v="ICICI"/>
  <tag k="amenity" v="atm"/>
</node>
```

```
<node id="2450211342" lat="28.4745865" lon="77.070711" version="2"
timestamp="2013-10-17T11:37:11Z" changeset="18401720" uid="56597" user="Oberaffe">
  <tag k="name" v="icici bank"/>
  <tag k="amenity" v="atm"/>
</node>
```

```
<node id="2399657393" lat="28.4249884" lon="77.0992852" version="1"
timestamp="2013-07-28T17:56:17Z" changeset="17129014" uid="47892" user="richlv">
  <tag k="amenity" v="atm"/>
  <tag k="operator" v="ICICI Bank"/>
  <tag k="opening_hours" v="24/7"/>
</node>
```

Now one can easily see that how for amenity **atm** same bank name is mentioned differently by different user. This is also true for so many other bank and atm names. Hence name **generalisation** was performed.

Similar **name unification** was performed on the gas station names having **amenity** as **fuel**.

▪ Bar and Pub ambiguity

Now for a finicky drinker, bar and pub may hugely different but for general population, **bar** and **pub** mean to serve the same thing; alcoholic beverages. Hence all the nodes having **amenity** as **pub** are replaced by **amenity** as **bar**.

▪ Problematic characters

There were many problematic characters and some of those listed below are cleaned. These are handled differently. Some are completely removed while others are merely modified like below

- Backslashes ('\\'): Removed the backslashes only
- Double Quotes ('\"''): Removed the quotes only
- Hindi names: removed the complete entry of tag-value
- Prefix with colons: replaced with part after the colon

MongoDB Statistics

▪ Basic Statistics on Database

a. Size of the file and Database

File Size Uncompressed: **184.1 MBs**
Total Database Size: **133.9 MBs**
Total Database Index Size: **8.4 MBs**
Avg. Document Size in Database: **142.6 Bytes**
Total number of Documents: **984,374**

b. Total number of ways, node, area

```
> db.openStreetDataMap.find({"type": "node"}).count()  
832455  
> db.openStreetDataMap.find({"type": "way"}).count()  
3330  
> db.openStreetDataMap.find({"type": "area"}).count()  
148569
```

c. Top 5 contributors

```
match = {"$match": {"created": {"$exists": 1}}}  
group = {"$group": {"_id": "$created.user", "total": {"$sum": 1}}}  
sort = {"$sort": {"total": -1}}  
project = {"$project": {"_id": 0, "name": "$_id", "total_conrtributions": "$total"}}  
limit = {"$limit": 5}  
pipeline = [match, group, sort, project, limit]
```

Rank	Name	Total Contributions
------	------	---------------------

1	anushap	70489
2	Naresh08	66777
3	sdivya	64030
4	hareesh11	60736
5	pvprasad	58621

d. Users contributed only once

```
> db.openStreetDataMap.aggregate([
... {"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
... {"$group": {"_id": "$count", "num_users": {"$sum": 1}}},
... {"$sort": {"_id": 1}},
... {"$project": {"_id": 0}}
... {"$limit": 1}
])
{"num_users": 107 }
```

e. Number of unique users

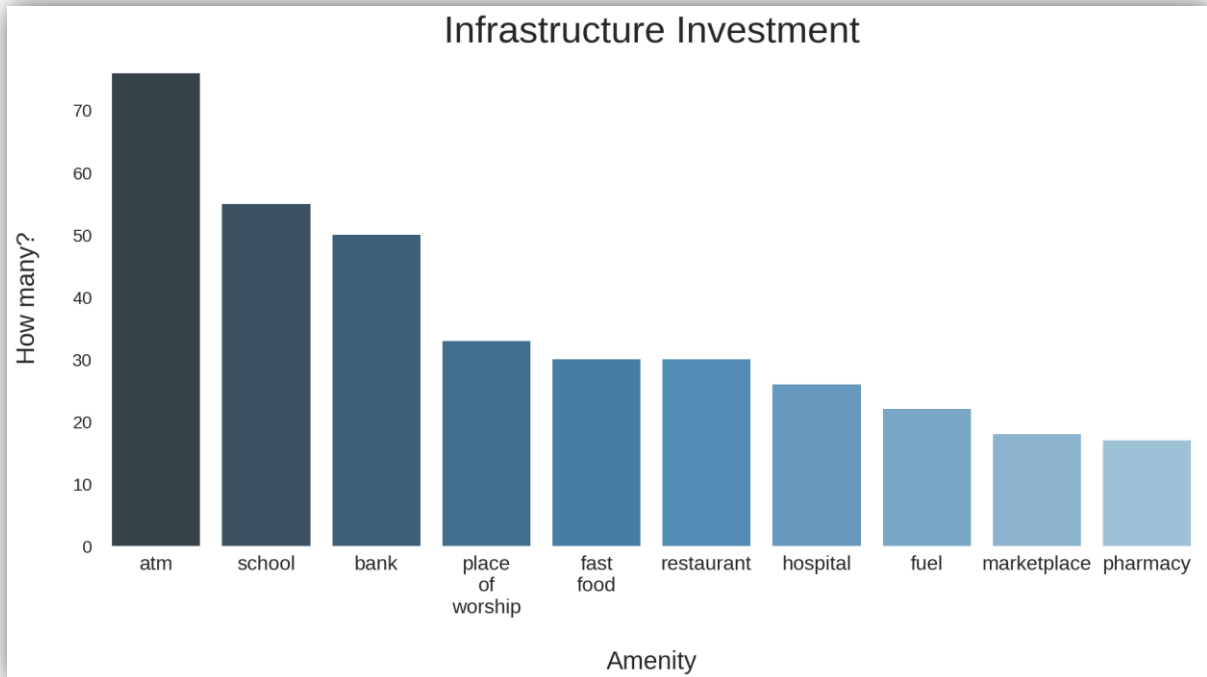
```
> db.openStreetDataMap.distinct('created.user').length
398
```

■ Additional Statistics on Database

a. How City Invested in Infrastructure?

The following MongoDB pipeline displays how city invested in infrastructure. Instead of displaying the output, the data is plotted and the following result is obtained.

```
match = {"$match": {"amenity": {"$exists": 1}}}
group = {"$group": {"_id": "$amenity", "total": {"$sum": 1}}}
sort = {"$sort": {"total": -1}}
project = {"$project": {"_id": 0, "amenity": "$_id", "count": "$total"}}
pipeline = [match, group, sort, project]
```

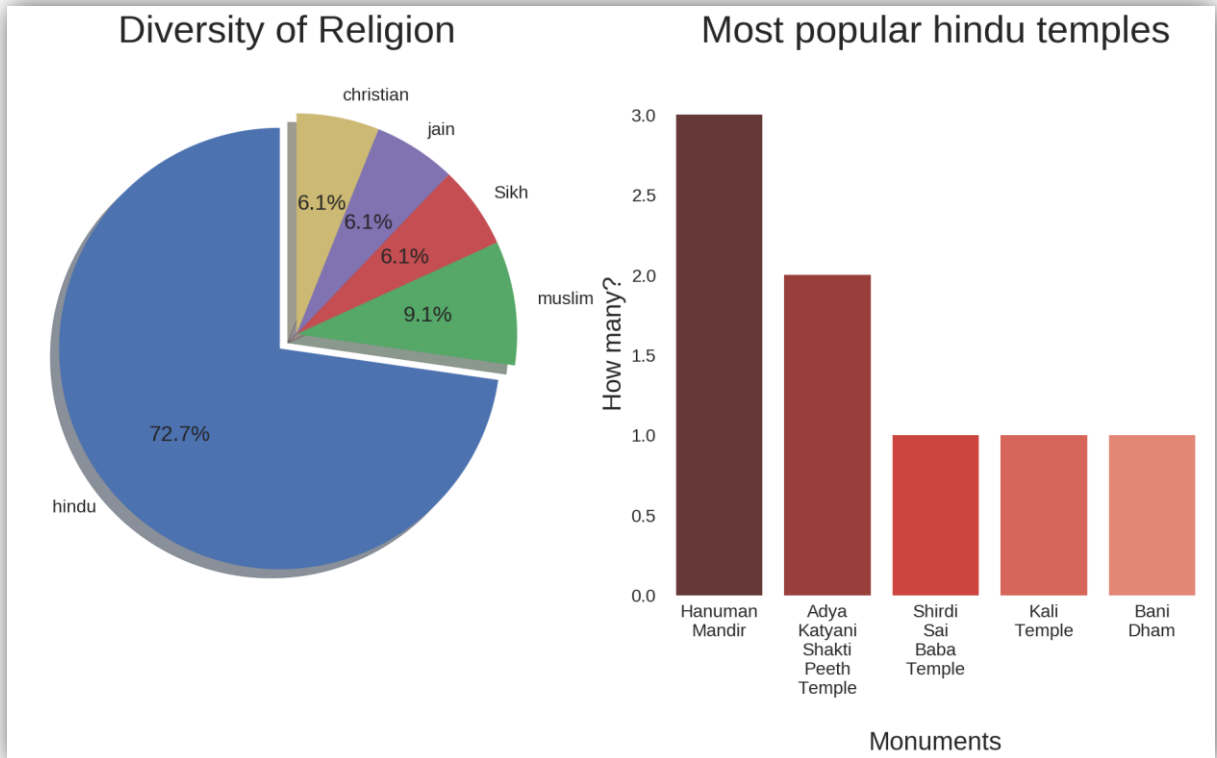
We can see that citizens in Gurgaon is more focused on money and career than they are in health. Moreover, even after so much of **banking** and **atm** spread, we can see that **marketplace** is comparatively less. This may indicate that people like to earn and spend less. Finally, we have more **place_of_worship** as compared to **hospital**. I guess it's a typical Indian story.

b. What is the diversity of the religion?

The Mongo query for this was the following but the result is interesting.

```
match = {"$match": {"amenity": {"$exists": 1, "$in": ["place_of_worship"]}}}
group = {"$group": {"_id": "$religion", "total": {"$sum": 1}}}
sort = {"$sort": {"total": -1}}
project = {"$project": {"_id": 0, "religion": "$_id", "count": "$total"}}
pipeline = [match, group, sort, project]
```

```
match1 = {"$match": {"amenity": {"$exists": 1, "$in": ["place_of_worship"]}}}
match2 = {"$match": {"religion": {"$exists": 1, "$in": ["hindu"]}}}
group = {"$group": {"_id": "$name", "total": {"$sum": 1}}}
sort = {"$sort": {"total": -1}}
project = {"$project": {"_id": 0, "monument": "$_id", "count": "$total"}}
pipeline = [match1, match2, group, sort, project]
```

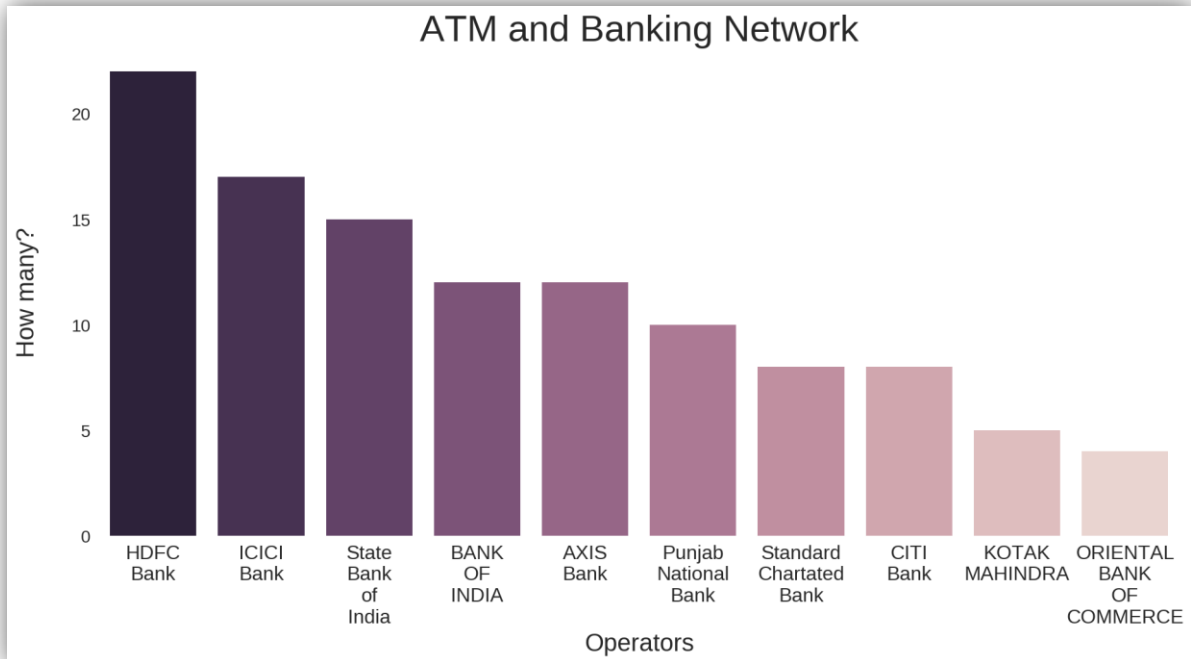


This was surprising but already known answer. Gurgaon is located in North India known to have high Hindu population as compared to other religion. This is also a known fact that people in North India worship **Hanuman** a lot and hence you can see that Hanuman temple is very popular Hindu worshipping place as compared to others.

c. Which bank has most ATM to banking network?

```
match1 = {"$match": {"amenity": {"$exists": 1, "$in": ["atm", "bank"]}}}
group = {"$group": {"_id": "$name", "total": {"$sum": 1}}}
sort = {"$sort": {"total": -1}}
project = {"$project": {"_id": 0, "bank_or_atm": "$_id", "count": "$total"}}
pipeline = [match1, group, sort, project]
```

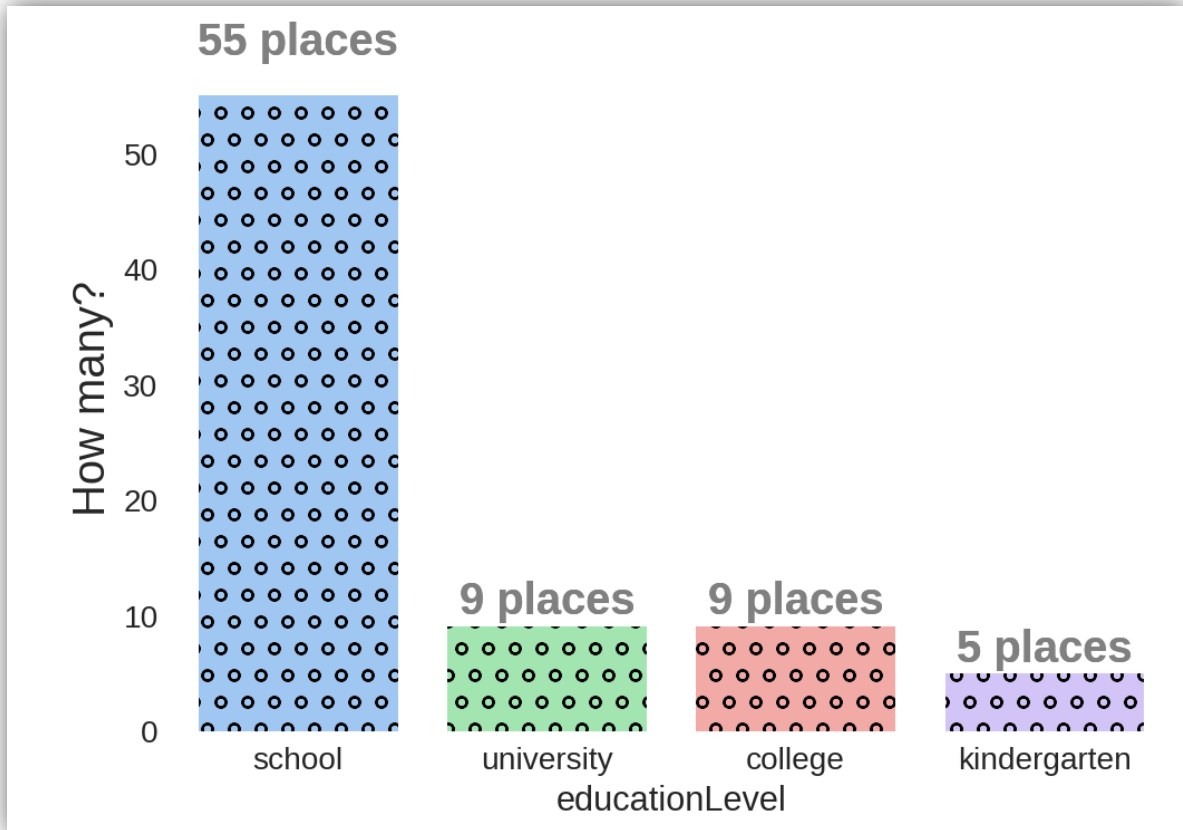
The above mentioned Mongo Query shows how banking and ATM network is spread over the city specially knowing that **bank** and **atm** is one of most popular infrastructure where city invested.



This is apparently true because **HDFC and ICICI** are two banks said to be the biggest competitor of each other and they have expanded quite a lot in last few years. They even beat the Govt bank **State Bank of India**, which shows govt needs to keep up now. One notable mention is **Kotak Mahindra Bank** which started few years back only and already expanded quite a lot to be in to 10 list.

d. How citizen focus on education?

```
match1 = {"$match": {"amenity": {"$exists": 1, "$in": ["school", "college", "university", "kindergarten"]}}}
group = {"$group": {"_id": "$amenity", "total": {"$sum": 1}}}
sort = {"$sort": {"total": -1}}
project = {"$project": {"_id": 0, "educationLevel": "$_id", "count": "$total"}}
```



What pattern this shows? Well this shows large number of schools compared to kindergarten. May be because parents are now more focused on home schooling at the early stage of the life and focus on primary/school education most.

The drastic fall in the number of colleges and universities shows less number of students are interested in higher studies in this region. This may be due to students preferring some better locations or may be college acceptance rate is low in this region.

Interestingly, we saw this region has largely invested on bank and ATMs which means this region has fairly high number of earning people. Now but college acceptance rate is low. So this may mean that this region has high number of immigrant workers.

Conclusion

Even after the cleaning of the data, the data cleaning is no way complete. The data cleaning could have been more intricate with better usage of **location** data or the **timestamp** data. But one thing is definitely apparent; the amount of intuitiveness required to explore and clean the data. Not only it requires the knowledge of the domain but it requires to know the data and

definitely a massive amount of time and energy. OpenStreetData was however much more streamlined but it could have been more difficult if data would have collected from different location with different formatting. Some OpenStreetData had some issues with **address** but this had fortunately none. We have use **cElementTree** module which is the **C** implementation of **ElementTree**. This ensures faster parsing of large xml data. The data can be improved by cleaning some unrelated information.

Anticipated Problem: For example, let's look at the following query

```
> db.openStreetDataMap.aggregate([  
..."$match": {"created_by": {"$exists": 1}}},  
... {"$group": {"_id": 1, total: {"$sum": 1}}},  
... {"$project": {"_id": 0}}  
])
```

```
{"total": 141}
```

This is an example of the unrelated data in the OSM file. Some nodes/ways are observed to have a tag called **created_by** which has nothing to do with the **user**. With little more inspection on the data, I have found that the value for the tag **created_by** is **JSOM**. With little more googling, I found that JSOM is nothing but Java OSM tool; a tool/software used to enter the node/way data. Now this information might be useful for one purpose only and that is if someone is interested to use the OSM data to get statistics on **which tool is contributing by how much?** But unfortunately, this data is not present in all the nodes except some few and most of them have value JSOM and hence that kind of statistics is not possible here. There is no other utility that can be found from this tag. The users may not have entered the data on their own. It's the tool which entered this data automatically.

Solution to the problem: This data can be easily cleaned from our cleaning module. We just need to check if there is any entry called **created_by**. If there is, then we reject that tag-value pair and move to the next one.

Difficulty in implementing Solution: But the above mention idea is only for a single tag-value pair i.e. {"created_by": "JSOM"}. It is needless to say that there is an immense amount of research and data auditing required to analyse the fact that if there is any irrelevant data or not like this type. This would immensely increase the development and throughput time.

Probable solution to solve it could be finding a pattern in the irrelevant data. If we can see the pattern, then it is easier to find the problem. Understanding the OSM data and its application usage can expedite the process of deciding the fact that if the data is relevant or irrelevant. This means a collective effort of finding irrelevant data by finding patterns and

understating of the application of the OSM data can decrease the time required to remove the irrelevant data quickly.

Finally, as people in this region possess smart phones, contributing to the OpenStreetData is very helpful and may solve most of the problems related to location and accuracy. OpenStreet data can be used to create some of the best application for amenity search in locality or by some government institutions to study the geography as we have done. This would help them create your city a better place.