1. Chosen Data and problems detected:

Map : https://www.openstreetmap.org/export#map=12/33.7526/-84.3525

I chose data of the Atlanta metropolitan area because I lived there for a while and I was curious if I could fin interesting facts and statistics from this city. At first glance I tried looking up the street names just like in the course to see if it was a good starting point for auditing the data. The street names where a bit messy, some being abbreviated and different types of abbreviation for the same word. Also found that the postal codes were inconsistent some having the state prepends to the postal code and some having extra digits at the end.

Other things I discovered:

I noticed that there were some entries that had multiple house numbers, at first I thought it was an error but I did some searching in the openstreetmap api and found that the multiple house number in one entry could mean several thing one being a shopping mall.

I also notice that under the building key there were entries with the value “Yes” instead of the actual building type Ex: School, Post office. But in the Api I discovered that this was an okey value if the user didn't know what kind of building it was.

Examples:

- •Street Names: (“S Tryon St Ste 105”)
- •postal Codes: (“GA30324”, “30313-1594”)

Importing Data into MongoDB

Once the MongoDB client was install in my linux operating system, I used the code provided in lesson 6 to get the data ready for the data base.

Code Snippet:

This code was used to get the right tags into the json file

```
def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :
        if 'visible' in element.attrib:
            node['visible'] = element.attrib['visible']
        node['created'] = {}
        for i in CREATED:
            node['created'][i] = element.attrib[i]
        node['pos'] = [float(element.attrib[i]) for i in element.attrib if i == 'lon' or i == 'lat']
        node['pos'].sort(reverse=True)
        node['type'] = element.tag
        node['id'] = element.attrib['id']
        node['address'] = {elem.attrib['k'][5:]:elem.attrib['v'] for elem in element.iter('tag') if
elem.attrib['k'] in attribs}
        if node['address'] == {}:
            node.pop("address", None)
        if element.tag == "way":
            node['node_refs'] = [elem.attrib['ref'] for elem in element.iter("nd")]
        return node
```

Then, this part was used to write the Json file

```
def process_map(file_in, pretty = False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return file_out
```

The data was written into a Json file which I imported directly into a MongoDB collection named city.

Correcting city addresses

In mongoDB I query the city's addresses and cleaned it up, so the abbreviated words would be complete. Example “Flat Shoals Avenue Southeast” was “Flat Shoals Ave SE”.
Code:
This checks for a street name containing Ave and changes it into Avenue
```
for street in  db.city.find({"address.street": {"$regex": r"Ave\b" }}):
        streetName = street[u'address'][u'street']
        street_type = "Avenue"
        name = re.sub(r"Ave\b", street_type, streetName)
        street[u'address'][u'street'] = name
        print name
        db.city.save(street)
```

Postal Codes

For the postal codes I decided to remove the prefix letters “GA” and remove the numbers after the “-”.
It would be much easier to deal with normal postal codes, also there weren't that many postal codes with this differences so its best to unify them anyways.
Examples:”30303” was “GA30303” or “30303-3224”
code example:

I wanted to figure out also, how many of this postal codes belong only to the city of atlanta and to see if the remaining postal codes were border line cities or just wrong codes.

First I looked up how many of the total records had postal codes:

```
db.city.find({"address.postcode": {"$exists": 1}}).count()
93890
```

Then I looked up atlanta postal codes:

```
db.city.find({"address.postcode": {"$regex": '^303'}}).count()
92303
```

Last I looked up all the other postal codes(even if I could just have subtracted):

```
 db.city.find({"address.postcode": {"$regex": /^(?!303)/}}).count()
```

```
1587
```

When I looked into this remaining postal codes I found that they are all from nearby cities:

Decatur → 30033
Mableton → 30126
Forest Park → 30294

2. Data Overview

In the data overview I will go over some basic information about the files, data and how I gathered this information.

File sizes

Atlanta.osm .... 388 MB
Atlanta.osm.json .... 441 MB

Number of documents

```
> db.city.find().count()
1812268
```

Number of nodes

```
> db.city.find({"type":"node"}).count()
1620161
```

Number of ways

```
> db.city.find({"type":"way"}).count()
192107
```

Number of unique users

```
> db.city.distinct("created.user").length
588
```

Top 1 contributing user

```
> db.city.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},{"$sort":
{"count": -1}},{"$limit":1}])

{ "_id" : "Saikrishna_FultonCountyImport", "count" : 1215258 }
```

3. Additional Ideas

looking at the atlanta.osm data I can see that there are still many interesting places that haven't been added to the map, also I looked at the map area of Pereira, Colombía where I currently live and notice this even more. Many of the users are bots and this may lead to the gap in information, many small businesses aren't appearing at all.

Possible solution: a possible solution to this lack of participation from real users and the lack of small-medium businesses appearing could be to develop a mobile app that can track the location of the user and prompt a little alert asking the user for input on an unrecognized area. This can lead to an everyday feed of data that can very well benefit and make a more detailed oriented map.

Possible problems: One would be the lack of motivation to get people interested enough to download an app to constantly feed it data. This could be settled by adding rewards and ranks. Another problem would be the quality of the data, given that the app means that people will usually use it in an everyday environment and possibly on the run, the data will tend to be sloppier that usual. For this, the app should have already established values and restrictions that allows users to enter data in its best possible format.

More interesting data:

Oldest entry:

```
 db.city.aggregate([{"$group":{"_id":"$created.timestamp"}},{"$sort":{"_id":1}}])

{ "_id" : "2007-10-13T01:31:01Z" }
```

Most recent entry:

```
db.city.aggregate([{"$group":{"_id":"$created.timestamp"}},{"$sort":{"_id":-1}}])

{ "_id" : "2015-04-21T06:09:46Z" }
```

Top appearing amenities

```
> db.city.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"_id":"$amenity",
"count":{"$sum":1}}}, {"$sort":{"count":1}}])

{ "_id" : "place_of_worship", "count" : 1113 }
{ "_id" : "parking", "count" : 923 }
{ "_id" : "parking_space", "count" : 837 }
{ "_id" : "restaurant", "count" : 382 }
{ "_id" : "school", "count" : 333 }
{ "_id" : "bicycle_parking", "count" : 161 }
{ "_id" : "fast_food", "count" : 119 }
{ "_id" : "hospital", "count" : 117 }
{ "_id" : "fuel", "count" : 112 }
{ "_id" : "bench", "count" : 79 }
{ "_id" : "cafe", "count" : 74 }
{ "_id" : "bank", "count" : 57 }
{ "_id" : "atm", "count" : 50 }
{ "_id" : "fire_station", "count" : 50 }
{ "_id" : "bar", "count" : 46 }
{ "_id" : "library", "count" : 45 }
{ "_id" : "post_office", "count" : 45 }
{ "_id" : "pub", "count" : 42 }
```

Biggest religions

```
> db.city.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"place_of_worship"}},

{"$group":{"_id":"$religion", "count":{"$sum":1}}},

{"$sort":{"count":-1}}])

{ "_id" : "christian", "count" : 332 }
{ "_id" : "muslim", "count" : 2 }
{ "_id" : "hindu", "count" : 1 }
```

Most popular cuisines

```
> db.city.aggregate([{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant"}},
{"$group":{"_id":"$cuisine", "count":{"$sum":1}}},{"$sort":{"count-1}}])

{ "_id" : "american", "count" : 46 }
{ "_id" : "burger", "count" : 42 }
{ "_id" : "pizza", "count" : 39 }
{ "_id" : "sandwich", "count" : 34 }
{ "_id" : "mexican", "count" : 31 }
{ "_id" : "coffee_shop", "count" : 26 }
{ "_id" : "chicken", "count" : 15 }
{ "_id" : "chinese", "count" : 11 }
{ "_id" : "regional", "count" : 11 }
{ "_id" : "italian", "count" : 9 }
{ "_id" : "seafood", "count" : 7 }
```

Conclusion

I can conclude that the openstreetmap map it's a great platform that inspires people to contribute to build a massive online map, but there are still a couple of upgrades that could be made to motivate and facilitate the data gathering. Many map regions are still very much incomplete and the already gathered data is missing some global structure, yet it's still very easy and fun to manage. Mongodb it's a great tool to store and query json files, very easy and straight forward queries don't take long at all to process. With the skills acquired in this project it will be much easier to start data collection and storing process, with easy data auditing and a great querying structure that brings some basic analysis into play.

References

MongoDB: http://docs.mongodb.org/master/#

Openstreetmap: https://wiki.openstreetmap.org/wiki/Main_Page