DATA WRANGLING PROJECT

Vítor Bernardes

OPEN STREET MAP DATA WRANGLING WITH MONGODB

Introduction

The objective of this project is to obtain, clean and examine an Open Street Map area data set. We will do so through the following steps:

1. Get Open Street Map data set for analysis
2. Verify the data set for potential problems
3. Clean problematic data — as much as possible programmatically
4. Convert the cleaned data into JSON for input into MongoDB
5. Examine our cleaned data set in MongoDB
6. Come up with potential ideas for improvement

Let us go through each of those steps in detail and document our process.

Getting the data set

The area chosen for our investigation is a large area comprising much of the southeastern part of the Brazilian state of Minas Gerais and a small part of the adjacent state of Rio de Janeiro. The biggest city in the area is called Juiz de Fora, and it is the city I currently live in.

That region does not contain any major metropolitan areas, which have more visibility and are not as much likely to have many data points that will require cleaning. Therefore, our area will potentially present several opportunities to clean the data.

The data was extracted using the Mapzen Metro Extracts tool and resulted in a **147.3MB** uncompressed XML file.

Checking the data set for problems

Upon examination of the data set, it was found that most user-entered text data was contained within *tag* elements. So a function was used to collect all unique key-value pairs from those elements, which were inspected in search of major cleaning opportunities.

The main problems and issues found in the data set that were addressed by this project include inconsistencies with:

- CEP (postcode) formatting
- City names
- Suburb names
- House numbers
- Street names
- Elevation values
- Opening hours
- Phone numbers

Let us quickly review each case and provide examples of problems found.

CEP formatting

CEP numbers are the Brazilian equivalent of post codes and must adhere to the following format: *nnnnn-nnn*. So correct examples would include 36770-000, 36025-275, and so forth.

Several CEP numbers were found in our data set with a thousands separator and without the dash (e.g. 36.205-276 and 36033340).

City names

Some city names in our data set contained state information and extra information not relevant to the field. Examples include “Chiador MG” and “Viçosa - MG”, where “MG” is the state abbreviation.

Suburb names

Suburb names did not present as much problems, however there were a couple of instances that contained extra spaces and inaccurate capitalization.

House numbers

There were some minor inconsistencies for house numbers in our data set, such as numbers with a thousands separator and inconsistent formatting for “s/n” values (which mean addresses without house numbers).

Street names

Street names posed a significant challenge, as there were several problems and inconsistencies found. Those ranged from typos and overly abbreviated names to extra spaces and capitalization issues.

A function was created to access the Brazilian Postal Services website and extract — using `BeautifulSoup` — official approved street types to be used when checking for problematic names.

Elevation

Some elevation values included a measurement unit, which prevent them from being correctly interpreted by Open Street Map.

Opening hours

Opening hour entries included several different user-entered formats, such as “06:00 a 22:00”, “06:00 às 17:00”, and “10:00 até as 15:00”. We will look to standardize those values into the following format: *nn:nn-nn:nn*.

Phone numbers

Finally, phone numbers also included differing formats, such as “+55 32 3572 1122”, “+55 32 3690-8327”, “222554-2298”, “32 33712568”, and “32-3216-5293”, among several others. The values will also be standardized, using the following format: *+nnnnnnnnnnn*.

Some entries were also found to contain invalid values, which will be ignored and not imported into the database.

Cleaning the data

After the problems with our data values were identified, a plan was designed to handle the inconsistencies present in each field specifically. All of the cleaning was done programmatically, via functions that manipulated the problematic values through several means, most notably regular expressions, so as to standardize their formatting and correct any issues found.

Here are some examples of cleaning that was performed on the problems found:

Type of data	Problematic value	Cleaned value
CEP	36301046	36301-046
City	Viçosa - MG	Viçosa
City	Barra de São Francisco (distrito)	Barra de São Francisco
House number	s/ nº	s/n
Street name	Estr. Mariana	Estrada Mariana

Street name	PRAÇA PADRE NELSON TAFURI	Praça Padre Nelson Tafuri
Street name	R. Arlindo Bruner	Rua Arlindo Bruner
Elevation	1860 m	1860
Opening hours	06:00 às 17:00	06:00-17:00
Opening hours	8:00 - 12:00 / 14:00 - 18:00 Seg a Sex	08:00-12:00 / 14:00-18:00 Seg a Sex
Phone number	(22) 2559-1141	+552225591141
Phone number	+55 (32) 3217 4034	+553232174034
Phone number	32-3216-5293	+553232165293

One code example can be found below. The following function was used to standardize opening hours:

```
def clean_opening_hours(hours):
    """Standardize format for opening hours."""

    one_digit_hour_re = re.compile('(?<!\d)(\d{1})(:\d{2})')
    two_hours_re = re.compile(u'(\d{2}:\d{2})[aàsátée \-]+
(\d{2}:\d{2})[Hh]?')

    def add_leading_zero(match_obj):
        return '0'+match_obj.group(1)+match_obj.group(2)

    def standardize_with_dash(match_obj):
        return match_obj.group(1)+'-'+match_obj.group(2)

    hours = one_digit_hour_re.sub(add_leading_zero, hours)
    hours = two_hours_re.sub(standardize_with_dash, hours)

    return hours
```

Data conversion

Once all functions for cleaning each specific identified field had been coded, the data set was processed by a script and mapped to a JSON structure, for insertion into a MongoDB database.

All three kinds of elements were processed — *nodes*, *ways*, and *relations* — and "nested" fields (which contained colons, e.g. *addr:street* and *addr:postcode*) were grouped within a container *address* field, with subfields for each corresponding value.

The resulting JSON file size was **177.6MB**.

Data examination using MongoDB

After the JSON file was created, it was imported into a local MongoDB instance with the following command:

```
mongoimport --db osm --collection southeast --drop --file ./Juiz\ de\ Fora\ Region.osm.json
```

The data was explored via a Python script using the PyMongo driver. The functions that made use of the MongoDB pipeline framework used a wrapper function to perform the actual call to the database.

Below are some examples of information extracted from the database.

Number of elements

```
pipeline = [ { '$group' : { '_id' : '$element_type',
                            'count' : { '$sum' : 1 } } },
             { '$sort' : { 'count' : -1 } } ]
```

The query above yielded the following results:

Element type	Number of entries
node	722602
way	51657
relation	1010

Number of cities

```
def count_unique_cities(db):
    return len(db.southeast.distinct('address.city'))
```

Using the Python function above, we learn there are **44** cities in this data set.

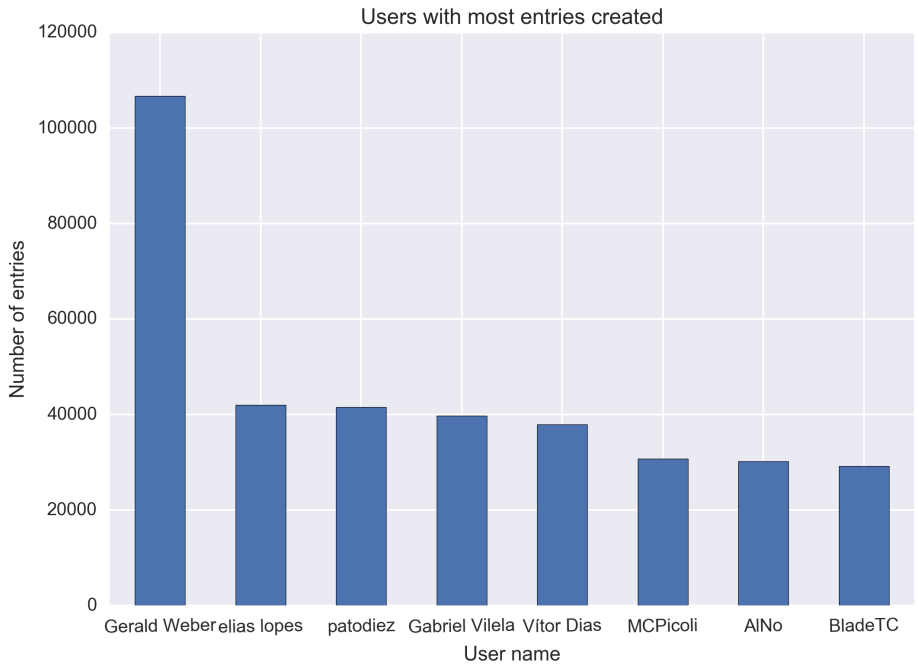
Number of unique users

```
def count_unique_users(db):
    return len(db.southeast.distinct('created.user'))
```

The Python function displayed above showed **399** unique users contributed to this data set.

Most frequent contributors

```
pipeline = [ { '$group' : { '_id' : '$created.user',
                            'count' : { '$sum' : 1 } } },
             { '$sort' : { 'count' : -1 } },
             { '$limit' : 8 } ]
```

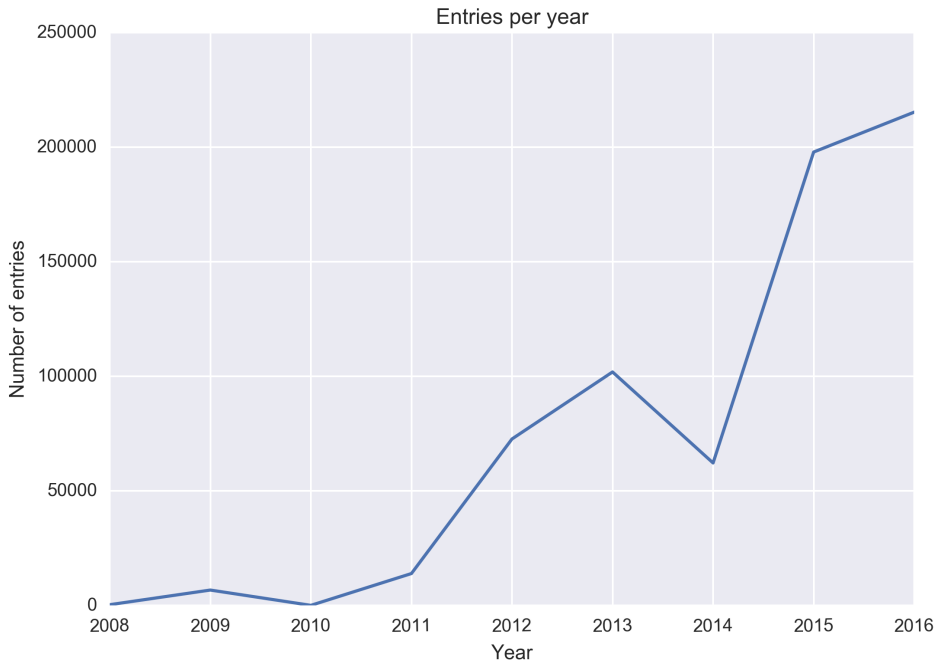


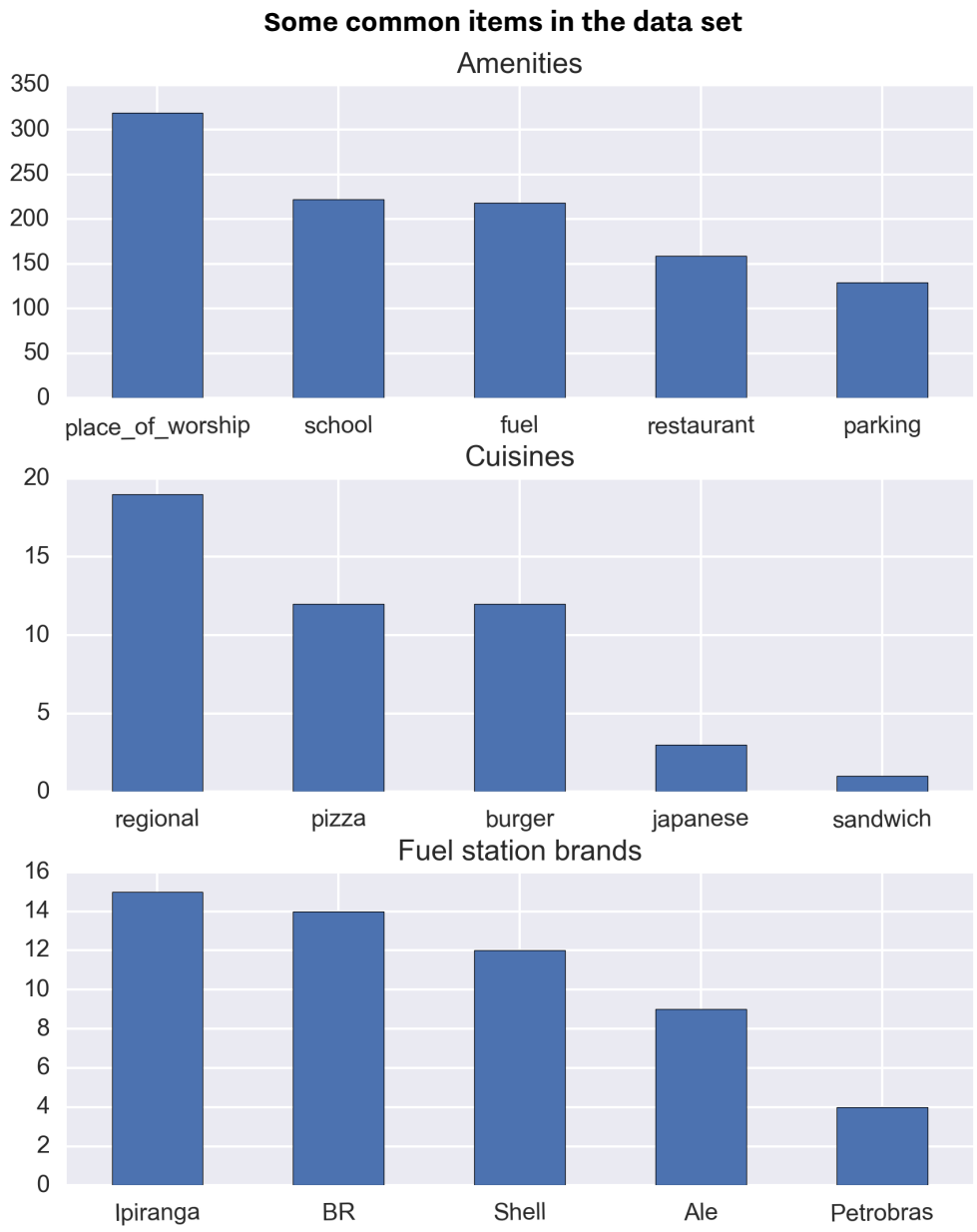
Number of entries per year

We were also able to plot how many entries were created for this data set in each year. First we retrieved from the database the timestamp for all entries, using:

```
db.southeast.find(projection={'created.timestamp':1, '_id':0})
```

Then the Python script extracted only the year from those values, which were represented in a Pandas series. The first and last years present in the data set (2007 and 2017, respectively) were removed from the plot so that only whole years worth of entries were taken into account.





Examples of common items found in the data set

These charts used data obtained from the database with the following functions:

```
def get_amenities(db):
    """Get the 5 most popular amenities."""
    pipeline = [ { '$match' : { 'amenity' : { '$ne' : None } } },
                 { '$group' : { '_id' : '$amenity',
                                'count' : { '$sum' : 1 } } },
```

```
                { '$sort' : { 'count' : -1 } },
                { '$limit' : 5 }
    ]
    return aggregate(db, pipeline)

def get_cuisines(db):
    """Get the 5 most popular cuisine types in the data set."""
    pipeline = [ { '$match' : { 'cuisine' : { '$ne' : None } } },
                { '$group' : { '_id' : '$cuisine',
                               'count' : { '$sum' : 1 } } },
                { '$sort' : { 'count' : -1 } },
                { '$limit' : 5 }
    ]
    return aggregate(db, pipeline)

def get_fuel_stations(db):
    """Get the 5 most popular fuel station brands."""
    pipeline = [ { '$match' : { '$and' : [ { 'amenity' : 'fuel' },
{ 'brand' : { '$ne' : None } } ] } },
                { '$group' : { '_id' : '$brand',
                               'count' : { '$sum' : 1 } } },
                { '$sort' : { 'count' : -1 } },
                { '$limit' : 5 }
    ]
    return aggregate(db, pipeline)
```

Ideas for improvement

Street names

The street names contained in the data set posed a significant challenge, due to the vastly different formatting employed by users when entering their submissions, not to mention the countless typos that are to be expected.

In regard to typos, a function was created to check for them on street names. First, we retrieved all street names from *way* elements representing streets in the database:

```
db.southeast.find({'element_type':'way', 'highway':'residential',
```

```
'name':{'$ne':None}},
                    projection={ 'name':1, '_id':0 })
```

Once those names were obtained, we compared them to each other to find similar values, which might be potential typos, or even incorrect user-entered entries for the same street name. A Python standard library function was used to evaluate the ratio of similarity between street names:

```
ratio = difflib.SequenceMatcher(lambda x: x == ' ', name,
name_to_compare).ratio()
```

The table below lists some examples of similar names found.

Street name	Similar street name
Rua Joemir de Farias	Rua Joemir Farias
Rua Francisco de Assis	Rua São Francisco de Assis
Rua Vicente Vieira da Mota	Rua Vicente Vieira Mota
Rua Honório Antônio da Silva	Rua Honorato Antônio da Silva
Rua Artur J Silva	Rua Artur Silva
Rua Sílvio Romero	Rua Silvio Romeiro
Rua Engenheiro Moura Neves	Rua Engenheiro Mauro Neves
Rua Independência	Rua Idependencia

The table indeed shows examples of typos and also misspellings of street names. That information could be used to further clean the street data.

One problem with the implementation of that check is that, for each street name, the whole list is compared to it. That algorithm is quadratic and a higher performance alternative should be investigated.

One other possible improvement when cleaning the data is to check the Brazilian Postal Services — arguably the authority on street names — database and extract the correct names for different street entries. Such a function could be used to correct typos or include missing information, such as a street type (street, avenue etc.) or even missing words.

A function was created to do exactly that. It takes a street name to check and the city the street is in, and returns the cleaned name obtained from the Brazilian Postal Services database. Here are a few examples of its use:

```
>>> print clean_name_from_postal('Dr. Romualdo', 'Juiz de Fora')
Rua Doutor Romualdo

>>> print clean_name_from_postal('Barao do Rio Branco', 'Juiz de
Fora')
Avenida Barão do Rio Branco
```

Standardizing other formats

Although some degree of cleaning was performed on several fields for this project, some of the data might benefit from further standardization as well, such as *opening hours*.

Conclusion

The original data set represents a region with no major metropolitan areas, containing data about 44 cities in the southeastern Brazilian states of Minas Gerais and Rio de Janeiro. Several steps were taken to audit the data, identify cleaning opportunities, come up with cleaning approaches and run the scripts created to clean the data based on those approaches.

Although, as we have discussed, there is still room for improvement and additional cleaning, I believe the data quality has been much improved and satisfactorily cleaned for the purposes of this project.