Udacity data Analyst Nanodegree
Project 2 – Data Wrangling with Mongo DB
Diego Menin
02/07/2015

# 1. Preparing the Data (and problems with it):

1) Choosing a map area

The map area chose was Dublin – Ireland, which is the city I currently live. The file was 228,903 Mb and contains 3,465,383 rows.

2) Sub setting the data:

Since the file is quite big, I used the *SourceData.py* to create a sample of the main file with 10% of the size. That sample file was used to develop and test the code that I used to analyse the whole file. All the following results and conclusions are reported on top of the complete file.

3) Pre-analysing the data

All the pre-analysis was done with the *PreAnalyseData.py* python code. It consists of a series of "checks" that either outputs the result of an analysis (on a text format) OR produces a file\update.
This file was not created to run all the checks on the same execution. Since each check is a "one time run" only, they are all commented on the main method so if you want to run any of them, all you have to do is uncomment the desired one. I didn't feel I should be spending time on making this process more interactive.

a. Check 1 – TAG counts:

The first check used iterative parsing to process the map file and find out not only what tags are there, but also how many, to get the feeling on how much of which data you can expect to have in the map:

```
{'bounds': 1,
 'member': 40445,
 'nd': 1341013,
 'node': 995574,
 'osm': 1,
 'relation': 2634,
 'tag': 689063,
 'way': 172015}
```

We can see that we have eight different tags; two of them are "osm" and "bounds", which represent the region itself and its boundaries, that's why we only have one of each for Dublin. I know for a fact (it is discussed on the course) that we are interested on the "tag" and "way" tags, so we'll focus on these and ignore the rest. It's beyond the scope of this work to explain the meaning of the other tags, but if you are interested in it, please go to this link.

b. Check 2 - Analyse the "k" value on the "tag" TAG:

The "way" and "node" tags have some base information about a specific location, but they also have subtabs with "key-value" pairs of interesting data. Here's an example of these tags referent to the "Brazilian Embassy" in Dublin (the file was cut to fit the document):

```
<node id="1731613977" lat="53.3335744" lon="-6.2637359" version="2" t
    <tag k="name" v="Brazilian Embassy"/>
    <tag k="amenity" v="embassy"/>
    <tag k="country" v="BR"/>
    <tag k="addr:city" v="Dublin"/>
    <tag k="addr:street" v="Charlotte Way"/>
    <tag k="contact:fax" v="+353 1 475 1341"/>
    <tag k="addr:country" v="IE"/>
    <tag k="contact:email" v="brasembdublin@brazil-ie.org"/>
    <tag k="contact:phone" v="+353 1 475 6000"/>
</node>
```

Since our goal is to insert data into MongoDB, we want to check the "k" value for each "<tag>" and see if they can be valid keys in MongoDB, as well as see if there are any other potential problems.
After we run the check, we can see that we don't have any major problem, only two tags that have a "+" sign in the end:

```
Check 2:
Element:
contact:google+
Element:
contact:google+
{'lower': 431712, 'lower_colon': 229158, 'other': 28191, 'problemchars': 2}
```

This will be dealt with on a later stage of the process.

c. Check 3 was only an exploratory one. The goal was to find out how many unique users have contributed to the Dublin map. I won't be outputting the full list here, only a subset:

```
Check 3:
set(['!i!',
     '0123456789',
     '12939',
     '4property',
     '4rch',
     '904473',
     '9thmeath',
     '@LS@',
     'AElderink',
     'ALE!',
     'AMB',
     'APLIE',
     'Acei',
     'Adam Lipsky',
```

d. Check 4 relates with street name data quality. It is a little more complex than what we've been doing so far but the ultimate goal is to have reliable and meaningful street names. This was an iterative process because I didn't know what challenges I was going to encounter so couldn't code for them ahead of time.

We are going to analyse the "addr:street" element and look for inconsistent street names. For example, we see two locations bellow, where on the first, the address ends with the word "street" and on the second, "Main St.", it end with "St.", which is an abbreviation for "Street":

```
<node id="571302907" lat="53.385888" lon="-6.374419" v
    <tag k="name" v="The Vineyard"/>
    <tag k="amenity" v="pub"/>
    <tag k="addr:street" v="Main Street"/>
</node>
<node id="571720156" lat="53.2022145" lon="-6.1104325"
    <tag k="name" v="Holland&#39;s Bar"/>
    <tag k="phone" v="+35312862448"/>
    <tag k="amenity" v="pub"/>
    <tag k="addr:street" v="Main St."/>
</node>
```

First of all, a list of "expected" values was created. It contains entries we are happy with it. Here is an initial example:

```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons"]
```

It just means that where we find addresses ending with the values above, we'll consider them correct and move on, so I ran a code that looks at all addresses on the file and outputs addresses that are note on the list.

At this moment the "iterative process" I mentioned started. Looking at the list, I had to decide which values were OK and which weren't.

Here are a few examples of the output and actions taken:

1. The "Main St." value we saw above:
    a. 'St.': set(['Main St.']),
2. Another misspell of the word "Street":
    a. 'Sreet': set(['Parliament Sreet']),
3. Another abbreviation for street, this time without the dot:
    a. 'St': set(['Aungier St', 'Thomas St']),
4. A perfect valid address, ending with "Terrace", which is only outputted because the word "Terrace" is not on the "expected" list above – so in this case, it should be added to the list:
    a. 'Terrace': set(['Aikenhead Terrace',
        'Albert Terrace',
        'Alexandra Terrace',
        'Arbour Terrace'…

The next step was to code something to replace the "bad" addresses with better ones. It was done using a "mapping" dictionary where I can say "map the string "St" to "Street". Here is an example of the dictionary and the "expected" function with the string "Terrace" added (more values were added during the analysis – this is the initial state of these variables):

```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons", "Terrace"]


mapping = { "St": "Street",
            "St.": "Street",
            "Sreet" : "Street",
            "Ave":"Avenue",
            "Rd.":"Road"
            }
```

And by calling the mapping function, I can output the following suggestions:

Main St. => Main Street
Spruce Ave => Spruce Avenue
Griffith Ave => Griffith Avenue
Thomas St => Thomas Street
Aungier St => Aungier Street
Strand Rd. => Strand Road

At the moment, they are only suggestion for the “interactive process” I mentioned. Once the process is done this function I’ll be plugged in to the main process as a clean-up step.

One fun thing to point out is that there are several addresses ending with the word “centre” and several others with the word “center”. Although there is no difference in meaning between “center” and “centre”, “center” is the preferred spelling in American English, and “centre” is preferred in varieties of English from outside the U.S. so I choose to make “centre” the official one by adding it to the mapping dictionary.

4) Relationship between “City”, “postal districts” and “residential area”:

The city of Dublin is composed by several “postal districts” prefixed by the word “Dublin” plus a number, for example: “Dublin 1”, “Dublin 2”, “Dublin 3” and so on. Inside most of the districts (but not all of them), we have what’s called a “residential area”; for example, “Ranelagh” is an area inside “Dublin 6”.

The way the data is today, the field "addr:city" is used to store one of these 3 different information. Sometimes it contains the word “Dublin”, sometimes the district and sometimes the postal code as we can see from the example bellow:

```
<tag k="addr:city" v="Dublin 8"/>
```

This is very hard to fix programmatically because we don’t know which of the 3 types of info is being captured on that field. What I did was to implement a simple rule to address this problem: If the string ends with a number, I assume that it is a postal code and add it to the “postal code” key and remove the number from the “addr:city” key, so for example, if I see “Dublin 8” as the city, this is how the JSON will look like:

```
{'address':
        {
         'city': 'Dublin ',
         'postal code': 'Dublin 8'
        } …

}
```

5) Generating JSON:

The goal of this step is to transform the data into JSON, so it can be inserted into mongo DB. The following rules were stablished:

- Process only 2 types of top level tags: "node" and "way"
- All attributes of "node" and "way" should be turned into regular key/value pairs, except:
- Attributes in the CREATED array should be added under a key "created"
- Attributes for latitude and longitude should be added to a "pos" array
- If second level tag "k" value contains problematic characters, it should be ignored
- If second level tag "k" value starts with "addr:", it should be added to a dictionary "address"
- If second level tag "k" value does not start with "addr:", but contains ":", process it same as any other tag.
- If there is a second ":" that separates the type/direction of a street, the tag should be ignored.

All the process was done in the function “process_map”, including the data fixes we’ve identified on previous processes. To run this function, is also necessary to run the street check to populate the “better_names” dictionary with the correction.

So here is an example of the function’s output ran on the Brazilian Embassy file I showed on Item 3b. **Please note that I added a dummy mapping from “Way” to “WayTest” just to exemplify the address clean up function working. This mapping won’t be part of the final run:**

```
{
  "amenity": "embassy",
  "fax": "+353 1 475 1341",
  "name": "Brazilian Embassy",
  "created": {
    "changeset": "12446579",
    "user": "mackerski",
    "version": "2",
    "uid": "6367",
    "timestamp": "2012-07-23T08:36:40Z"
  },
  "country": "BR",
  "pos": {
    "0": 53.3335744,
    "1": -6.2637359
  },
  "email": "brasembdublin@brazil-ie.org",
  "phone": "+353 1 475 6000",
  "address": {
    "city": "Dublin",
    "street": "Charlotte WayTest",
    "country": "IE"
  },
  "type": "node",
  "id": "1731613977"
}
```


6) Inserting into MongoDB:

mongoimport -d project -c dublin --file dublin_ireland.json

# 2. Data Overview

The following data overview was developed by running queries on the "Dublin" collection. I will present it at a textual format where, after each claim, I'll include a reference to the query used to produce it. All the queries will follow this section and the data quality will be dealt with on the next section.

The data loaded consist of 1.087.158 documents (Q1). These documents were originally classified as "nodes" or "ways" (995.489 nodes and 91.669 ways – Q2 and Q3). The overall Dublin city mapping was done with the contribution of 1006 distinct users (Q4). From all those users, we can highlight the following top 5 ones with their correspondent number of contributions:

{ "_id" : "Nick Burrett", "total" : 215930 }
{ "_id" : "mackerski", "total" : 176058 }
{ "_id" : "Dafo43", "total" : 143640 }
{ "_id" : "brianh", "total" : 101732 }
{ "_id" : "Ignobilis", "total" : 54217 }

Since I used the Brazilian Embassy to exemplify the data wrangling process, I thought that it would be a good exercise to check how many other embassies we've got on the dataset (37 - Q6) - and which are they (Q7). We've got a total of XXX embassies; here are 5 of them:

{ "name" : "Embassy of the Republic of Kenya", "country" : "KE" }
{ "name" : "Japanese Embassy", "country" : "JP" }
{ "name" : "Embassy of Austria", "country" : "AT" }
{ "name" : "Embassy of France", "country" : "FR" }
{ "name" : "Embassy of the Republic of Poland", "country" : "PL" }

The Dublin amenities:

Dublin is a multicultural city. If you are looking for a good meal, you'll find a diverse range of cuisines, the top 5 present are (Q8):

```
{ "_id" : null, "total" : 255 }
{ "_id" : "italian", "total" : 49 }
{ "_id" : "indian", "total" : 38 }
{ "_id" : "chinese", "total" : 30 }
{ "_id" : "pizza", "total" : 21 }
```

But of course, the Irish are wildly known by its famous beer, the Guinness, and by been quite heavy drinkers. This can be confirmed by the number of bars and pubs on the city: 653 (Q9)

But not only of food and drinks the Irish live, there several other leisure activates available in the city. (Q11) We have 739 parks, 185 sports centres (plus 18 stadiums), 106 playgrounds for the kids, 89 golf courses and 48 gardens.

**Queries used:**

- Q1 - Number of collections:

```
db.dublin.count()
```

- Q2 - Number of "nodes":

```
db.dublin.find({"type":"node"}).count()
```

- Q3 - Number of "ways":

```
db.dublin.find({"type":"way"}).count()
```

- Q4 - Number of unique contributors:

```
db.dublin.distinct("created.user").length
```

- Q5 - Top 5 contributors:

```
db.dublin.aggregate([
        { $group: { _id: "$created.user", total: { $sum: 1 } } } ,
        { $sort: { total: -1 } },
        { $limit:5}
])
```

- Q6 - Number of Embassies:

```
db.dublin.find({"amenity" : {"$regex" : "embassy"}}).count()
```

- Q7 - List of embassies:

```
db.dublin.find({"amenity" : {"$regex" : "embassy"} } , { name: 1, country: 1, _id:0}  ).pretty()
```

- Q8 – Top 5 cuisines:

```
db.dublin.aggregate([
        { $match : {amenity:{$exists:1}, amenity:"restaurant"}}
        ,{ $group : {_id:"$cuisine", total:{$sum:1}}}
        , { $sort: { total: -1 } }
        ,{ $limit:5}
])
```

- Q9 – Number of Bars and Pubs:

```
db.dublin.find({"amenity" : {"$regex" : "[Pp]ub|[Bb]ar" }}).count()
```

- Q10 – Number of Amenities:

```
db.dublin.find({"amenity" : {"$exists" : 1}}).count()
```

- Q11 – Leisure options:

```
db.dublin.aggregate([
        { $match:{"leisure":{"$exists":1}}},
```

```
{ $group:{"_id":"$leisure",total:{"$sum":1}}},
{ $sort: { total: -1 } }
,{ $limit:10}])
```

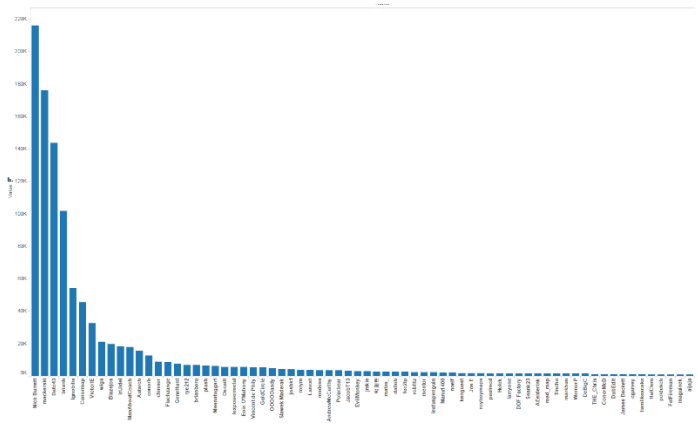
# 3. Additional Ideas:

Even though, the data collected is quite extensive, while analysing it, I could easily conclude it is incomplete. For example, while analysing the embassies, I could find 37 entries on the dataset, while a quick research on the Dublin embassy website (http://www.dublin.info/embassies/) shows 63 – so there is clearly room for improvement when it comes to completeness of the data. Another example regarding completeness was found analysing the "most common" restaurant (by cuisine) in Dublin. The dataset only shows one Brazilian restaurant and I know for a fact that there are at least 4 of them.

Regarding data quality, we could also see on the "top cuisine" in the city is "null". By checking one of them on the open street map website, I could see that this information was just not provided, meaning that there are a lot of "restaurants" without "cuisine" – so this is a good point to consider when it comes to data quality. Also, I believe that cuisine "pizza" should probably be classified as "Italian", otherwise we'd need to have "sushi" instead of Japanese for example.

**Other Suggestions to Open Street Maps to improve overall data quality:**

- The number of contributions per user is extremely skewed, meaning that there are only a handful of users that actively contribute to the project (at least on this city). A user score would be a good indication on how reliable a particular contribution is; this wold be especially useful on the data cleaning process, where we could spend more time on the user's posts with less experience.



  Also, a mechanism to encourage users to contribute to OpenStreetMaps would be very interesting and give the dataset more diversity.

- I don't know if it falls outside OpenStreetMaps' objective but it would be good to expand it by giving the possibility of adding notes or maybe reviews to a place or establishment.
- It would be very useful to be able to use google maps API to get more detailed information (or verify current information). Maybe something programmatically could be implemented by cross checking latitude and longitude between the systems.
- As I discussed before, some fields should be mandatory when others specified. For example, by specifying "restaurant" on amenity, user must inform the "cuisine".

**Code:**

The Python Files mentioned on this project can be found on:

https://github.com/dmenin/UdacityDataAnalystNanoDegree/tree/master/Mongo/UdacityNanoDegreeP2/FinalProject/