1 PROBLEMS ENCOUNTERED WITH THE DATA

I noticed three sources of problems in the data for the city of Madrid, after running the audit.py program.

1. Street types and names
2. Postcodes
3. Phone numbers

1.1 PROBLEMS WITH STREET TYPES AND NAMES

When I first run the audit.py program, I wanted to check from the values of the "addr:street" attribute, the street types of the data, that were not in a list (expected), which I had previously created, containing common street types (calle, avenida…). (These are in Spanish but they are the equivalent to street, avenue … in English). After running the program, I saw that there were some valid types of street types that I had not included in the expected list, but were valid names for a type of street (travesia, camino…). Consequently, I decided to add those street types to the list of expected street types.

```
expected = ['calle', 'avenida', 'plaza', 'parque', 'glorieta', 'paseo', 'ronda',
            'camino', 'carretera', 'bulevar', 'cuesta','pasaje', 'travesia', 'urbanizacion', 'autovia',
            'barrio', 'ciudad', 'sector', 'via', 'cuesta', 'bajada', 'puerta', 'rinconada', 'callejon', 'poligono', 'senda',
            'costanilla']
```

In addition to that, I also encountered that there were some names of street types that were abbreviated (cr, av …) from valid names for types of streets. In order to create a consistent database, I decided that the names of street types had to all be in the same form (full words). Thus, I created a dictionary of key value pairs, where the key is the street type I wanted to change, and the value the right one. I called this dictionary corrections.

```
corrections ={'av': 'avenida', 'av.': 'avenida', 'call': 'calle', 'ctra': 'carretera', 'ctra.':'carretera', 'ctra,':'carretera',
            'crta':'carretera', 'crta.': 'carretera', 'carrterera':'carretera', 'pasage':'pasaje', 'avda.':'avenida', 'avda':'avenida',
            u'travesía': 'travesia', u'callejón':'callejon', 'rcda':'rinconada', 'corredera':'calle',
            'carrera':'calle', u'urbanización':'urbanizacion', u'autovía':'autovia', 'urb.':'urbanizacion', u'vía':'via',
            u'pol\xef\xbf\xbd\xef\xbf\xbdgono':'poligono', u'calleja/callej\xef\xbf\xbd\xef\xbf\xbdn':'callejon',
            u'prolongación': 'calle', 'cr': 'carretera', 'gran': 'calle'}
```

I also encountered problems with Unicode encoding for some of the street types. Due to Spanish having accented words, inconsistencies were encountered, where some users had inputted the same street types such as "travesía" with the accented "i" and others with a simple "i". The problem arose because the word with the accented letter had Unicode encoding and the other did not, and hence the audit.py considered "travesia" and "travesía" different street types. Thus I decided that for the databases all street types should be without accents and hence I added those cases to the corrections dictionary.

Another problem was that some streets were highways which start with an alphanumeric code. This alphanumeric code is formed by a letter followed by "-" and a number (M-30, A-1). Thus, the audit.py returned a different street type for every highway code. I wanted all highways to have the same street type; "highway", and hence I created a regular expression

(roads_re) to match these cases. I would then use this regular expression to identify these cases and create a “highway” street type for them in the clean.py program.

roads_re=re.compile(r'(([a-z]|[A-Z])\-[0-9]+)')

Finally, the last problem with the street types was that some streets were missing one. The street type was obtained with the audit.py program by parsing the first word of the street name in the xml file. This is because in Spanish street types always appear at the start of the sentence (‘Calle Maria de Molina’, where calle is the street type). However some street names were missing a valid street type at the start of the sentence (‘Maria de Molina’), and hence the audit.py considered that the street type was the first word of the street name. For these cases, a dictionary called modifications was created, such that the key was the first word of the street name and the missing street type the value. These dictionaries and lists were then used as a blueprint for the data cleaning process done with the clean.py program.

```
modifications = {'santa':'calle', 'virgen':'calle', 'san':'calle', 'ermita':'calle', 'daoiz':'calle', 'francisco': 'calle',
                 'salvador': 'calle', u'josé':'calle', u'fermín':'calle', 'antonio':'calle', 'rafaela':'calle',
                 'camilo': 'calle', 'fuencarral': 'calle', 'goya': 'calle', 'ventura': 'calle', 'sierra':'calle',
                 'tenerife':'calle', u'constitución':'calle', 'ginebra': 'Paseo', 'lina':'calle', 'real': 'calle',
                 'campezo': 'calle', 'veredilla': 'calle', 'aguado': 'calle', 'venezuela':'calle', 'bucarmanga':'avenida',
                 'fundidores': 'calle', 'paloma': 'calle', u'amnistía':'calle'}
```

1.2 PROBLEMS WITH POSTCODES

Two problems were encountered with the postcodes. Firstly, some postcodes were not from Madrid. In Spain the first two digits of the 5 digit postcode correspond to the province. Hence, all areas in the same province have the same two digits in their postcodes. For Madrid, these two digits are 28. However, some postcodes in the data did not start with 28. This problem was recorded because data elements with a postcode not from Madrid would have to be discarded when producing the JSON file to then import into MongoDB. A regular expression was produced to identify the valid postcodes (postcode_re). Secondly, some postcodes started with a capital “E” for Espana (Spain), followed by a valid 5 digit code. A regular expression (correct_postcode_re) was also created to identify these postcodes, which had to be corrected into the standard 5 digit code in the cleaning process. These regular expressions were then used in the clean.py program to clean and standardise the postcode data.

postcode_re = re.compile(r'^[2][8][0-9]{3}\b')

correct_postcode_re = re.compile(r'^[E][2][8][0-9]{3}\b')

1.3 PROBLEMS WITH PHONE NUMBERS

Firstly, it was found that some phone numbers had errors in them, such as missing a number or having non numeric characters like letters. These phone numbers were discarded. Secondly, some phone numbers were two different numbers separated by a slash or comma. These were split in

clean.py and then validated individually. Finally, most phone numbers showed inconsistencies in their format. Some were preceded by +34 (prefix for Spain), others by 0034, others by +0034 and others were not. It was decided that for the database phone numbers would either start by +34 followed by the number, or it would just be the number itself (without prefix). In addition to that, there were also inconsistencies between the number of spaces between digits. Some phone numbers had a space after some of the digits (94 56 78 345), and others had some of the digits separate by "-" (94-5678345). It was decided that all phone numbers would not have neither spaces between digits nor "-". These numbers were corrected in the clean.py program.

2 DATA OVERVIEW

Files:

- madrid_spain.osm ---------- 890533 KB
- madrid_spain.osm.json --------973258 KB

After following the query.py program, which contained all the queries to run in MongoDB, the following results were produced.

##db.command('dbstats')

```
{u'storageSize': 351977472.0, u'ok': 1.0, u'avgObjSize': 245.63241162082693, u'd
b': u'test', u'indexes': 1, u'objects': 4554891, u'collections': 1, u'numExtents
': 0, u'dataSize': 1118828861.0, u'indexSize': 41263104.0}
```

It can be seen that the number of documents is 4554891, and that the size of the data is 1118828861 bytes, which is equal to 1092606 KB.

len(db.Madrid.distinct('created.user'))
This query returned the number of unique users

db.Madrid.find({'type':'node'}).count()
This query returned the number of nodes

db.Madrid.find({'type':'way'}).count()
This query returned the number of ways

db.Madrid.find({'amenity':'shop'}).count()
This query returned the number of shops in madrid

The results for these queries can be seen in the picture below

```
number of unique users 2614
number of nodes: 4043802
number of ways: 511089
number of shops: 142
```

Additional queries were also run to find out about the rarest amenities, most popular cuisines, number of single contributing users and the greatest contributing users.

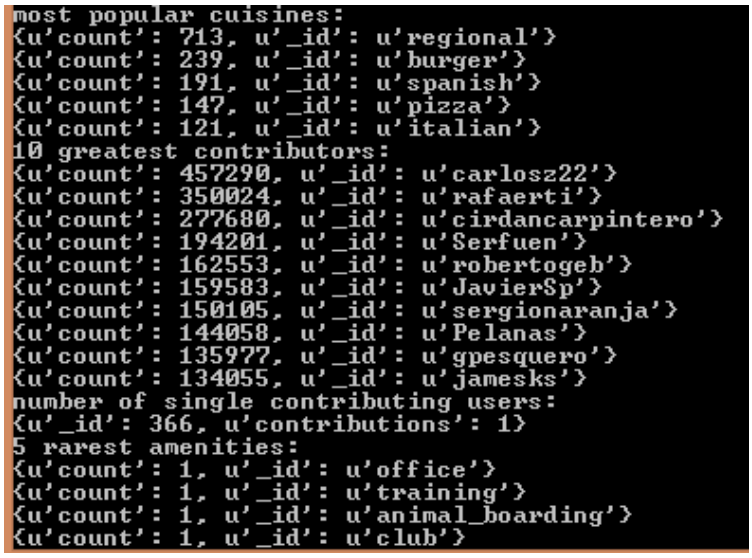
db.Madrid.aggregate([{'$group':{'_id':'$created.user', 'count':{'$sum':1}}}, {'$sort':{'count':-1}},{'$limit':10}])
This query returned the 10 greatest contributors to the data.

##db.Madrid.aggregate([{'$group':{'_id':'$created.user', 'count':{'$sum':1}}},
{'$group':{'_id':'$count', 'contributions':{'$sum':1}}}, {'$sort':{'contributions':1}}, {'$limit':1}])
This query returned the number of single contributing users

##db.Madrid.aggregate([{'$match':{'amenity':{'$exists':1}}}, {'$group':{'_id':'$amenity', 'count':{'$sum':1}}}, {'$sort':{'count':1}}, {'$limit':5}])
This query returned the 5 least popular (rarest) amenities in Madrid

##db.Madrid.aggregate([{'$match':{'cuisine':{'$exists':1}}}, {'$group':{'_id':'$cuisine', 'count':{'$sum':1}}}, {'$sort':{'count':-1},{'$limit':5}])
This query returned the 5 most popular cuisines in Madrid

These final queries produced the following results:



3 OTHER IDEAS ABOUT THE DATASET

Because the data was produced by several users, there are cases where some data attributes (cuisines, amenities …) contain different words that are synonyms, hence meaning very similar things. This is clear from the results of the most popular cuisines, where regional and Spanish both appear, and represent the same type of food. Hence the data for cuisines and amenities could be audited to produce a general scheme of clear distinct values to avoid values with similar meaning. This would be very beneficial in doing data analysis to obtain general patterns of data, but you would also detail in each data element. Also, queries that use group by cuisine or amenities could be quicker since they need to process less groups. There is, however, a problem with the interpretation of the schema. Some users might think that Italian cuisine includes pizza for example, and others might think of pizza as a completely independent thing. Hence, when developing the schema it is

important to provide information as to what each type of cuisine entails, and also to avoid including a large group (pizza) into another large group (Italian) because a lot of detail could be lost otherwise.

Open street map could limit users from inputting data values with accents. The benefit of doing this is that uniformity is obtained across the data since some users tend to write the data without accents, regardless of correct spelling, and others write data with accents (the way you spell it). This would also improve the analysis and processing of the data because differences between accented words and non-accented are not produced. The problem arises when in some languages the accents are indeed the means of differentiation between otherwise equal words. In Spanish, for example, accents are useful in differentiating verbal tenses of the same word. Thus, depending on the type of data and analysis, this implementation could be done. For this dataset, there is no examination of verbal tenses and hence the risk of equalising different words is very low.