Udacity Data Wrangling Project

Mark Wright

Choice of Location: Birmingham, UK *https://www.openstreetmap.org/relation/162378*

1. Problems Encountered In Map

Data volume:

birmingham_england.osm 1.15 GB

Since the input file was so large i firstly created a sample (1/100th) so that I could work with it more quickly and generated some simple counts for the types of tags contained within it. I assumed that this should be pretty representative and I could expect approximately 100* the amount of each of these in the total dataset.

In my sample:

```
{'node': 49158, 'other': 0, 'relation': 60, 'way': 8196}
```

I decided to only process the node and way tags, as we did in the exercise to problem set 6, and the data would be sufficient for the analysis I wanted to do. If I had more time I would have audited the relation elements contents to see what other value I could have derived.

Problem Characters

I wanted to audit the keys in tag elements to ensure there would be no problem parsing and loading the data I audited using regular expressions:

```
{'problemchars': 0, 'lower': 21141, 'other': 1537, 'lower_colon': 5794}
```

So I expect very little with problem characters in the full dataset, which means that most of the key value pairs should be ok to load without being dropped.

Data Consistency and Accuracy:

I wanted to understand a little more about the keys, particular which were the most prevalent and thus which i should spend more time on so I audited them.

The top 15 tag keys were:

```
source: 5999, building: 4860, highway: 2096, addr:street: 1904, name: 1473, addr:housenumber: 1318, addr:postcode: 1101, addr:city: 888, barrier: 813, created_by: 494, leisure: 369, natural: 359, landuse: 315, surface: 295, amenity: 285
```

I decided to focus on addr:street, addr:postcode, amenity, leisure, natural and landuse because I would use these in my further analysis later. I wanted to look at the values of some of these in more detail to see if I should create some mappings (like we did with street previously) to make the values consistent, so that any further analysis using the values was more accurate.

I also found there were a number of tags that had been created inaccurately be users, for example: piste:difficulty, piste:type. To my knowledge there are no skiing pistes in Birmingham! I thought that this may have been added as a joke and wondered if I should consider skipping any entry added by that user. I created an XPath to find the entry and found the user "robw". I then scanned all entries added by the user, the others seemed normal so I decided not to filter out the content. This led me to do a bit more research, and I found that there is indeed a dry ski slope!
http://www.ackers-adventure.co.uk/

Landuse

The landuse field was diverse so I decided to reduce down to a simple model (in a new field called land_sum so not to lose detail) so that we could infer some conclusions about different areas of the city. (see visualisation section below)

Postcode

Since I was going to use postcode in my analysis later I wanted to ensure that the field was well formatted and adhered to the standard UK format.
http://stackoverflow.com/questions/378157/python-regular-expression-postcode-search

```
r'[A-Z]{1,2}[\dR][\dA-Z]? \d[A-Z]{2}'
```

This showed that there were a number of postcodes that did not match the format. Some were incomplete e.g. ['B33', 'B38 8B'] some did not contain a space between the city descriptor and local descriptor e.g 'B297HW'
Since we could still do analysis over incomplete postcodes, for example show me all of x in postcode B38, I decided to keep the incomplete ones, as long as the first part was valid *r'[A-Z]{1,2}[\dR][\dA-Z]'*.
I decided to add a space into the postcodes that were missing them.

Nested keys

When processing my data, I decided that for all keys in tag nodes, if they contained a nested key eg. a:b I would write my parser to automatically nest the data. When trying to write the nesting code I hit problems with this element:

```
{'name': 'War Memorial', 'created': {'changeset': '20105323', 'user': 'FrViPofm', 'version': '5', 'uid': '107257', 'timestamp': '2014-01-20T15:17:39Z'}, 'listed_status': 'Grade II', 'source': 'visual survey;bing', 'historic': 'memorial', 'heritage': '2', 'type': 'node', 'id': '2291033100'}
```

The tag "heritage" was already defined with a string value, and when I tried to add a nested variant

```
{'k': 'heritage:operator', 'v': 'English Heritage'}
```

as a nested value under heritage, python complained because you cannot assign an object to a string. My solution whenever this was the case was to add a new element "v" to assign the value to, thus, this became:

```
{'heritage':{'v': '2', 'operator' : 'English Heritage'}}
if key_arr[0] in node and type(node[key_arr[0]]) is not dict:
 v = node[key_arr[0]]
 node[key_arr[0]] = {}
 node[key_arr[0]]['v'] = v
```

2. Data Overview

This section contains an overview of the data.

File overview

original file size: birmingham_england.osm 1.15 GB
processed file size: birmingham_england_osm.json 1.27GB

file imported into mongo using mongo import:
mongoimport --db users --collection osm.birmingham --type json --file /Users/Tommy/Dev/data_wrangling/assignment/data/birmingham_england_osm.json

Document Overview:

Total:

```
> db.osm.birmingham.find().count()
5735332
```

Nodes:

```
> db.osm.birmingham.find({"type": "node"}).count
4914897
```

Ways:

```
> db.osm.birmingham.find({"type": "way"}).count()
819397
```

User Stats:

Distinct Users:

```
>db.osm.birmingham.distinct("created.user").length
1786
```

Top 5 Contributing Users:

```
 >db.osm.birmingham.aggregate(  {'$group' : { "_id" : "$created.user", count : {"$sum" : 1}}}, {"$sort" : {"count" : -1}}, {"$limit" : 5});

{ "_id" : "brianboru", "count" : 2591152 }
{ "_id" : "blackadder", "count" : 486506 }
{ "_id" : "Miked29", "count" : 485221 }
{ "_id" : "mrpacmanmap", "count" : 156344 }
{ "_id" : "Curran1980", "count" : 138076 }
```

See visualization section below.

Other Stats

Number of bars

```
> db.osm.birmingham.aggregate({"$match" :{"amenity":{"$exists": 1}, "amenity" : "bar"}},{'$group' : {
"_id" : null, count : {"$sum" : 1}}});
{ "_id" : null, "count" : 136 }
```

Number of woods

```
> db.osm.birmingham.aggregate({"$match" :{"natural":{"$exists": 1}, "natural" : "wood"}},{'$group' : {
"_id" : null, count : {"$sum" : 1}}});
{ "_id" : null, "count" : 10183 }
```

3. Additional Ideas

Indexing for location analysis

In order to make some of the analysis I wanted to do on the data more efficient I decided to add some indexes. On the postcode field and the lat long pos field.

```
db.osm.birmingham.createIndex({'address.postcode' : 1})
db.osm.birmingham.createIndex( { 'pos' : "2dsphere" } )
```

What is the breakdown of cuisines in central Birmingham (B1)?

```
> db.osm.birmingham.aggregate({"$match" :{"address.postcode":{"$exists": 1}, "amenity" : {"$exists"
: 1}, "address.postcode" : {"$regex" : "B1.*"}, "amenity" : "restaurant" }},{'$group' : { "_id" : "$cuisine",
count : {"$sum" : 1}}})
{ "_id" : "indian", "count" : 1 } { "_id" : "thai", "count" : 1 } { "_id" : "italian", "count" : 1 }
{ "_id" : "french", "count" : 1 } { "_id" : "moroccan", "count" : 1 } { "_id" : "pizza", "count" : 1 }
{ "_id" : "caribbean", "count" : 1 } { "_id" : null, "count" : 32 } { "_id" : "greek", "count" : 1 }
```

Where is the nearest bank from my current location?

Supposing i am in central Birmingham lat 52.48, long -1.9, where is my nearest bank?

```
>db.osm.birmingham.aggregate({"$geoNear" : {"near" : { "type" : "Point", "coordinates" : [52.48,
-1.9]}, "distanceField" : "dist", "spherical" : "true"}}, {"$match" : {"amenity" : "bank"}});
```

Only returned one result, I was expecting a few more entries in the results, I did some analysis into the data and found that plenty of the nodes about banks had a lat, lon associated with them:

```
> db.osm.birmingham.find({"amenity" : "bank", "pos" : {"$exists" : 1}}).count()
204
```

I worked out after a while it was to do with the number of elements passing through the aggregation pipeline, the first stage was limiting this to 100, and thus only 1 of the 100 closest things was a bank! This solved the issue:

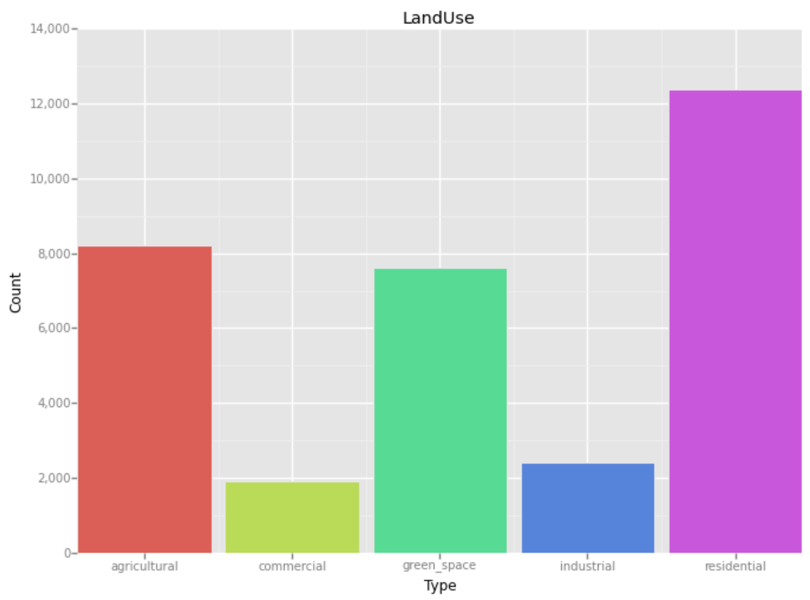
```
> db.osm.birmingham.aggregate({"$geoNear" : {"near" : { "type" : "Point", "coordinates" : [52.48,
-1.9]}, "distanceField" : "dist", "spherical" : "true", num : 100000}}, {"$match" : {"amenity" : "bank"}},
{"$project": { "name" : 1, "dist" : 1, "_id" : 0}}, {"$limit" : 5});
{ "name" : "Nationwide", "dist" : 54.47117581436675 }
{ "name" : "Coutts", "dist" : 114.3231353241792 }
{ "name" : "Chelsea Building Society", "dist" : 119.78433281182595 }
{ "name" : "Yorkshire Building Society", "dist" : 155.7486679875165 }
{ "name" : "Citysave", "dist" : 163.16630671995046 }
```

Visualisation

I wanted to use some of the techniques I learnt previously in the course to produce some visualizations of the aggregated data, so that I could more easily show trends.

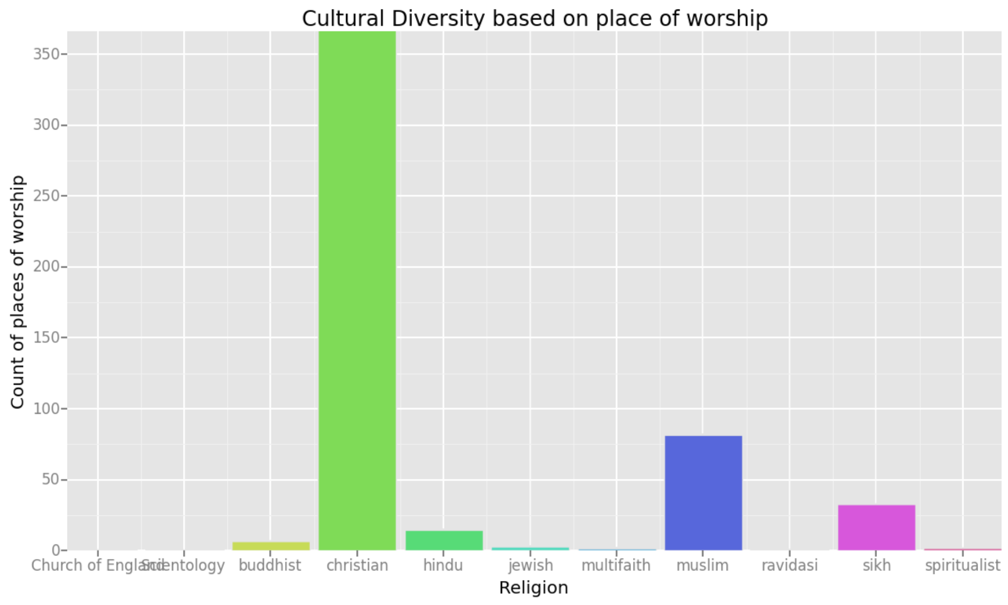
Land use overview based on land_sum summarisation

```
> db.osm.birmingham.find({"land_sum" : {"$exists" : 1}}).count()
32511
```



Because its hard to tell the granularity of things people categorised with land_use and its not based on geographical density this may be misleading.

Cultural diversity overview

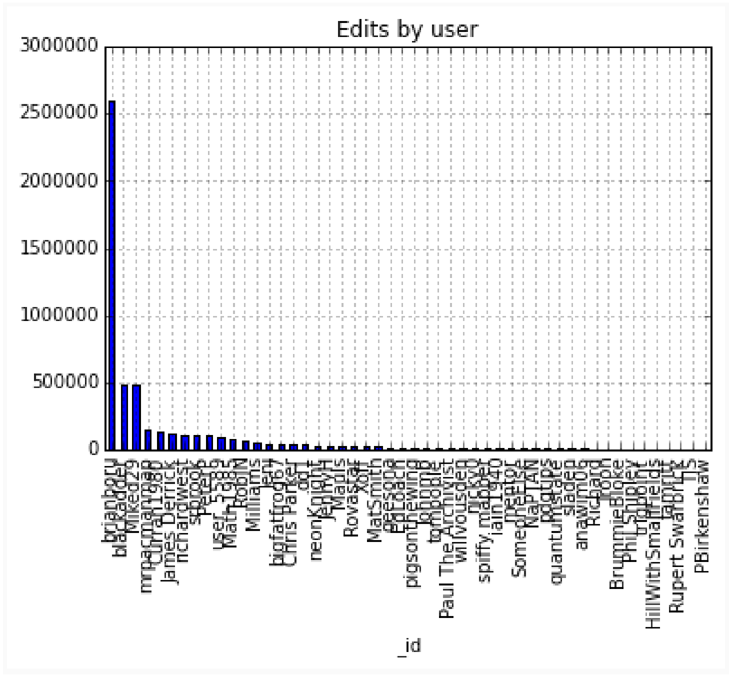


note: christianity had a total count of 1534, this is a zoomed in version so that other counts are more visible.

I would have liked to summarize the data by vicinity (based on city), however when I performed some simple counts on the data is showed that lots of the place_of_worship elements did not have that attribute. Pos (lat, lon) was much more complete.

```
> db.osm.birmingham.find({"amenity": "place_of_worship"}).count()
1913
> db.osm.birmingham.find({"amenity": "place_of_worship", "address.city" : {"$exists" : 1}}).count()
81
> db.osm.birmingham.find({"amenity": "place_of_worship", "pos" : {"$exists" : 1}}).count()
539
```

What is the distribution of user edits?



You can see that the user edits distribution is positively skewed with few people making the most edits. (This is only a subset of the top 50 contributors, this does not affect the distribution) Follow on work.

Follow on work

In follow on work I would split the city into 1km squared grids and use the position elements to group nodes by, we could then produce heat maps of the different vicinities to bring out certain facets such as, cultural make up, cusine types, places with the most bars etc.

Conclusion

When choosing to analyse different facets of the data, it was important to understand how complete the dataset was, and the best way to group data that you wanted to analyse. Using some characteristics of the data to infer position (such as postcode and city) were not as complete as lat long position, and thus using lat-long was a much better way to group by location.