DATA WRANGLING PROJECT

ANDRÁS SOMI, 2017. JULY

Data

- `budapest_hungary_inner.osm` (107 MB)
- `dump.json` (114 MB)

I created a custom Metro extract on Mapzen for the inner parts of Budapest, capitol of Hungary. The available basic extract of Budapest area was way bigger than this exercise requires (close to 1GB) and also contained areas that in reality are not part of the city (that may cause conflicts in postcodes or street name duplications).

I live in the city, I have a general understanding of the naming conventions, special characters, etc. Also the different language makes it difficult to simply reuse code snippets from the examples, so I need to thoroughly think through every piece of it.

Source: https://mapzen.com/data/metro-extracts/metro/budapest_hungary/

Auditing data

First, to have a general understanding of the data I ran through the whole xml counting the occurences of different attributes by tags and the overall count of the different tags. The dataset seems to be pretty uniform, ie. the different type of tags tend to have the same set of attributes for all the instances. For `node` tags `uid` and `username` attributes are missing in a tiny fraction of the cases.

```
TAG AND ATTRIBUTE COUNTS:

{'bounds': {'attributes': {'maxlat': 1, 'maxlon': 1, 'minlat': 1,
'minlon': 1},
            'count': 1},
 'member': {'attributes': {'ref': 68919, 'role': 68919, 'type':
68919},
            'count': 68919},
 'nd': {'attributes': {'ref': 547248}, 'count': 547248},
 'node': {'attributes': {'changeset': 422629,
                         'id': 422629,
```

```
                          'lat': 422629,
                          'lon': 422629,
                          'timestamp': 422629,
                          'uid': 422611,
                          'user': 422611,
                          'version': 422629},
           'count': 422629},
 'osm': {'attributes': {'generator': 1, 'timestamp': 1, 'version':
1},
          'count': 1},
 'relation': {'attributes': {'changeset': 7098,
                             'id': 7098,
                             'timestamp': 7098,
                             'uid': 7098,
                             'user': 7098,
                             'version': 7098},
              'count': 7098},
 'tag': {'attributes': {'k': 402982, 'v': 402982}, 'count':
402982},
 'way': {'attributes': {'changeset': 75148,
                        'id': 75148,
                        'timestamp': 75148,
                        'uid': 75148,
                        'user': 75148,
                        'version': 75148},
          'count': 75148}}
```

But most of the interesting data is stored in `tag` tags as key-value pairs in the `k` and `v` attributes. Let's look into these.

Auditing street names

For a 'gold standard' of types of public places I used the information from Wikipedia. This seems to be a complete, official list of the Hungarian names for different types of streets, roads and other public areas.

It's a pretty long list but most of the datapoints should end with one of the three most popular types (street, road, square). (In Hungarian we just put the type after the name of the street in small caps like 'Ilka *utca*' or 'Döbrentei *tér*')

I found 18 names that don't fit into any of the official categories, but most of them actually make sense as they are grammatical variations of basic types (eg. *'útja'* means the road of someone or something, so absolutely valid). The rest are some unique places like a castle (yes, there are castles in Budapest!) with their unique names.

```
UNEXPECTED STREET NAMES:

{'Erzsébet királyné útja',
 'Ferenciek tere',
 'Hadak útja',
 'Harminckettesek tere',
 'Ifjúság útja',
 'Kucsma',  # THIS IS INCORRECT
 'Kunigunda útja',
 'Magyar tudósok körútja',
 'Margitsziget',
 'Nagy Lajos király útja',
 'Népliget',
 'Népliget aluljáró',
 'Rákos-patak',
 'Rákospalotai körvasútsor',
 'Rózsák tere',
 'Vajdahunyadvár',
 'Vigadó téri hajóállomás',
 'Árpád fejedelem útja'}
```

We have only one entry that is suspicious: *'Kucsma'*. That's actually the name of the street not the type, and apparently *'utca'* (street) is missing from the end. *'Kucsma utca'* occures seven times in the dataset.

The search also brought up a few cases where the street name starts with lower case letter. This should be also handled before uploading the dataset to a database.

```
...
 'Zöldmáli lejtő': 7,
 'dessewffy utca': 1,  # SMALL CAPS
 'podmaniczky utca': 1, # SMALL CAPS
```

```
 'szabolcs utca': 1, # SMALL CAPS
 'százados út': 1, # SMALL CAPS
 'Ábel Jenő utca': 4,
...
```

Auditing postcodes
In Hungary we use four-digit postcodes. All Budapest postcodes start with 1 and the second and third digit denotes the number of district. There are 23 districts in the city, so the inner two digits should be between 01 and 23, except for the island called Margitsziget, where the inner two digits are 00.

There are no residential areas on the island but there are some cultural and sports facilities so we should still expect a few 00s to pop up in the dataset. (The Districts of Budapest (Wikipedia))

Four odd postcodes popped up in the audit. In the last one 'H' denotes Hungary in an international postcode format. It's a pattern we can easily correct programatically in the dataset.

`1503` and `1507` are most likely typos for `1053` and `1057` (confirmed by the street names) while `1476` seems to be a valid postcode even though it does not seem to adhere to the standard format (but Google also gives valid results)

```
{'1476': {'count': 1, 'tags': ['Üllői út']},
 '1503': {'count': 1, 'tags': ['Kérő utca']},
 '1507': {'count': 1, 'tags': ['Irinyi József utca']},
 'H-1026': {'count': 2, 'tags': ['Pasaréti út', 'Pasaréti út']}}
```

Auditing coordinates
I audited lattitude and longitude coordinates to be float numbers around 47.5 and 19 respectively. Not surprisingly no odd coordinates popped up thanks to the way the data was obtained.

Querying the data

Number of documents

```
> db.budapest.find().count()
```

```
504875
```

Count by type of tag

```
> db.budapest.aggregate([
    {'$group': {
        '_id': '$type',
        'count': {'$sum': 1}
    }},
    {'$project': {
        '_id': 0,
        'type': '$_id',
        'count': 1
    }}
])

[{'count': 7098, 'type': 'relation'},
 {'count': 75148, 'type': 'way'},
 {'count': 422629, 'type': 'node'}]
```

Top 5 postcodes by number of occurence

The 11th district is one of the biggest one (maybe the biggest one), no surprise that `x11x` postcodes are the most frequent in the dataset.

```
> db.budapest.aggregate([
    {'$match': {
        'address.postcode': {'$exists': True}
    }},
    {'$group': {
        '_id': '$address.postcode',
        'count': {'$sum': 1}
    }},
    {'$sort': {'count': -1}},
    {'$limit': 5},
    {'$project': {
        '_id': 0,
        'postcode': '$_id',
        'count': 1
    }}
```

```
])

[{'count': 1922, 'postcode': 1112},
 {'count': 1566, 'postcode': 1118},
 {'count': 1035, 'postcode': 1124},
 {'count': 1025, 'postcode': 1025},
 {'count': 735, 'postcode': 1089}]
```

Number of contributing users

Not too many compared to the population of 2 million people... `python`

```
> db.budapest.distinct('created.user').length
1232
```

Top 3 users

igor2 is the big winner with 3 times more entries than the second most active user.

```
> db.budapest.aggregate([
    {'$group': {
        '_id': '$created.user',
        'count': {'$sum': 1}
    }},
    {'$sort': {'count': -1}},
    {'$limit': 3},
    {'$project': {
        '_id': 0,
        'username': '$_id',
        'count': 1
    }}
])

[{'count': 134944, 'username': 'igor2'},
 {'count': 33466, 'username': 'MartinHun'},
 {'count': 23452, 'username': 'vasony'}]
```

Number and type of amenities

Apparently Budapest is the city of benches.

```
> db.budapest.find({'amenity': {'$exists': True}}).count()
```

```
10271

> db.budapest.aggregate([
    {'$match': {
        'amenity': {'$exists': True}
    }},
    {'$group': {
        '_id': '$amenity',
        'count': {'$sum': 1}
    }},
    {'$sort': {
        'count': -1
    }},
    {'$limit': 5},
    {'$project': {
        '_id': 0,
        'type': '$_id',
        'count': 1
    }}
])

[{'count': 1394, 'type': 'bench'},
 {'count': 975, 'type': 'restaurant'},
 {'count': 893, 'type': 'bicycle_parking'},
 {'count': 791, 'type': 'parking'},
 {'count': 695, 'type': 'waste_basket'}]
```

Most popular cuisines

Italian is our choice after the local tastes. (I guess pizza delivery is still a big business here.)

```
db.budapest.aggregate([
    {'$match': {
        'amenity': 'restaurant',
        'cuisine': {'$exists': True}
    }},
    {'$group': {
        '_id': '$cuisine',
        'count': {'$sum': 1}
    }},
```

```
    {'$sort': {'count': -1}},
    {'$limit': 3},
    {'$project': {
        '_id': 0,
        'cuisine': '$_id',
        'count': 1
    }}
])

[{'count': 76, 'cuisine': 'regional'},
 {'count': 61, 'cuisine': 'hungarian'},
 {'count': 54, 'cuisine': 'italian'}]
```

Further ideas

Dates and timestamps

It would be useful to have the timestamps in date format in MongoDB, but Python `datetime` objects cannot be serialized to JSON so in this workflow of importing the data at once into MongoDB we shouldn't change the timestamps from strings to dates.

```
entry = db.budapest.find_one({'created.timestamp': {'$exists':
True}})
timestamp = entry['created']['timestamp']
print(isinstance(timestamp, str))
True
```

There might be several ways to handle this:

1. Before creating the JSON file transform string timestamps to UNIX timestamps and store them as integers. This makes somewhat easier to create date-based queries and still doesn't break the JSON dump (still not the most convenient way to handle timestamps).
2. After importing the JSON data to MongoDB run a script that transforms the string timestamps to proper `datetime` objects and updates the appropriate field document-by-document. This would be a time-consuming operation but then we can use the timestamps as dates in queries and aggregations.