Udacity P3 Project

November 8, 2016

1 Udacity Project P3: Wrangle OpenStreetMap Data

1.1 Introduction

I used an OpenStreetMap export of the area of Paderborn, Germany, the place where I live.

1.2 Data audit and inconsistencies

Before converting the OpenStreetMap XML file to a JSON, I audited the data for inconsistencies.

Street adresses The street adresses looked fine to me. German street have a large variety of endings. The most common is "Straße" and "Weg". However, there are countless more and sometimes they are written with a "-", sometimes with a space and sometimes as a single word, so it's nearly impossible to check whether this is a valid name or not. I checked if there were any abbreviations but couldn't find any in the dataset.

Postocdes Then I checked the postcodes: all postcodes should have 5 digits and start with 3 or 5, which is correct for Paderborn and the surrounding areas.

```
In [26]: import os
         import xml.etree.cElementTree as ET
         from collections import defaultdict

         osmfile = "paderborn.osm"
         osmjsonfile = "paderborn.osm.json"

         for _, element in ET.iterparse(osmfile):
             if element.tag == "way":
                 for tag in element.iter("tag"):
                     if tag.attrib['k'] == "addr:postcode":
                         if len(tag.attrib['v']) != 5:
                             print(tag.attrib['v'])
                         if tag.attrib['v'][0] != "3" and tag.attrib['v'][0] != "5'
                             print(tag.attrib['v'])
```

```
85053
```

So there is obviously one wrong postcode in the data. However, when I checked the entry I found that this is the address of a chainstore where someone entered the address of the central office.

Phone numbers The phone numbers were really inconsistent. Some had the country code for Germany, so they were starting with "+49" or "0049". The continuation after this code was also inconsistent: some included the following "0", some did not. There were also some mobile phone numbers (beginning with 017..., for example) and some service numbers (starting with 0800...).

I also noticed that there were some "emergency_access_point" in the dataset which had set the phone number "112", so in order to leave these numbers as they are, I also added that.

```
In [ ]: def update_phone(phone_number):
            # http://stackoverflow.com/questions/6116978/python-replace-multiple-st

            rep = {"-": "", "(": "", ")": "", "/": "", " ": "","-": "", "+": "", "
            rep = dict((re.escape(k), v) for k, v in rep.iteritems())
            pattern = re.compile("|".join(rep.keys()))
            phone_number = pattern.sub(lambda m: rep[re.escape(m.group(0))], phone_

            if phone_number[:2] == "49":
                 phone_number = "0" + phone_number[2:]

            if phone_number[:4] == "0049":
                 phone_number = phone_number[4:]

            if phone_number[:2] == "00":
                 phone_number = phone_number[1:]

            if phone_number[0] != "0" and phone_number != "112":
                phone_number = "0" + phone_number

            return phone_number
```

1.2.1 Cuisines

Some restaurants also have some information about the cuisine they offer. I noticed some inconsistencies regarding the spelling or formatting. There were also four entries with German descriptions. Furthermore, there were some values with several cuisines seperated by semicolons, so I split them, checked every single value and converted them back to a string again.

```
In [45]: expected_cuisines = ["asian", "thai", "regional", "ice_cream", "chinese",
                              "turkish", "german", "pizza", "mexican", "italian",
                              "indian", "steak_house", "argentinian", "greek", "russ
                              "kebab", "french", "burger", "oriental", "sausages",
                              "cake", "shisha", "spanish", "donuts", "vietnamese",
                              "lahmacun", "tapas", "sandwich", "pasta", "salad", "su
                              "japanese", "balkan", "international"]
```

```
        cuisine_mapping = {
                            "Wok und mehr": "asian",
                            "Wurst": "sausages",
                            "Balkan und internationale Speisen": "balkan; internat
                            "icecream": "ice_cream",
                            "german and french": "german; french",
                            "Kuchen,_Flammkuchen,_Espresso,_Capppuccino,Getränke":
                            "cake; tarte flambée; coffee; drinks"
                            }

        def update_cuisine(cuisine):
            if cuisine not in expected_cuisines:
                if cuisine in cuisine_mapping:
                    cuisine = cuisine_mapping[cuisine]
                elif "," in cuisine:
                    cuisine = cuisine.replace(",", ";")
                if ";" in cuisine:
                    cuisine_list = cuisine.split(';')
                    for i in xrange(len(cuisine_list)):
                        if cuisine_list[i] in cuisine_mapping:
                            cuisine_list[i] = cuisine_mapping[cuisine_list[i]]
                    cuisine = ";".join(cuisine_list)
            return cuisine
```

1.3 Explore the data

1.3.1 Overview

```
In [46]: from pprint import pprint
         from pymongo import MongoClient

         client = MongoClient('localhost:27017')
         db = client["udacity"]
         paderborn = db.paderborn_osm

In [47]: print("Filesizes:")
         print('The original OSM file is {} MB'.format(os.path.getsize(osmfile)/1.0
         print('The original JSON file is {} MB'.format(os.path.getsize(osmjsonfile
```

```
Filesizes:
The original OSM file is 310.451692 MB
The original JSON file is 333.523074 MB
```

```
In [48]: print("Users: {}".format(len(paderborn.distinct("created.user"))))
```

```
Users: 906
```

```
In [49]: print("Number of documents: {}".format(paderborn.find().count()))
```

```
Number of documents: 1479054
```

```
In [50]: print("Number of nodes:",paderborn.find({'type':'node'}).count())
         print("Number of ways:",paderborn.find({'type':'way'}).count())
```

```
Number of nodes: 1266754
Number of ways: 212194
```

1.4 Further exploration

1.4.1 Top 10 leisure

```
In [51]: leisure = paderborn.aggregate([{'$match': {'leisure': {'$exists': 1}}},
                                         {'$group': {'_id': '$leisure',
                                                     'count': {'$sum': 1}}},
                                         {'$sort': {'count': -1}},
                                         {'$limit': 10}])
         print(list(leisure))
```

```
[{'_id': 'pitch', 'count': 557}, {'_id': 'playground', 'count': 486}, {'_id': 'par
```

1.4.2 Top 10 amenities

```
In [52]: amenity = paderborn.aggregate([{'$match': {'amenity': {'$exists': 1}}},
                                         {'$group': {'_id': '$amenity',
                                                     'count': {'$sum': 1}}},
                                         {'$sort': {'count': -1}},
                                         {'$limit': 10}])
         print(list(amenity))
```

```
[{'_id': 'bench', 'count': 4031}, {'_id': 'parking', 'count': 1676}, {'_id': 'wast
```

1.4.3 Benches

Funny enough, the most found amenity are benches. There are quite a lot of lakes here, all surrounded by waling and bicycle paths, so this is somehow reasonable. A quick look into the data revealed that there is even more information about these benches:

Material

```
In [53]: bench_material = paderborn.aggregate([{'$match':{'amenity': 'bench'}},
                  {'$group':{'_id':'$material',
                             'count':{'$sum':1}}},
                  {'$sort':{'count':-1}},
                  {'$limit':3}])

         print(list(bench_material))
```

```
[{'_id': 'wood', 'count': 2203}, {'_id': None, 'count': 845}, {'_id': 'metal', 'cou
```

Colour

```
In [54]: bench_colour = paderborn.aggregate([{'$match':{'amenity': 'bench'}},
                  {'$group':{'_id':'$colour',
                             'count':{'$sum':1}}},
                  {'$sort':{'count':-1}},
                  {'$limit':3}])

         print(list(bench_colour))
```

```
[{'_id': 'brown', 'count': 2119}, {'_id': None, 'count': 957}, {'_id': 'green', 'co
```

Seats

```
In [39]: bench_seats = paderborn.aggregate([{'$match':{'amenity': 'bench'}},
                  {'$group':{'_id':'$seats',
                             'count':{'$sum':1}}},
                  {'$sort':{'count':-1}},
                  {'$limit':3}])

         print(list(bench_seats))
```

```
[{'_id': '3', 'count': 2642}, {'_id': None, 'count': 968}, {'_id': '4', 'count': 14
```

1.4.4 Restaurant Opening hours

After information about benches was so detailed, I also checked the information about opening hours of resturants in order to find out if there were opening hours available and if yes, if they were just some kind of placeholder or detailed.

```
In [40]: restaurant_opening = paderborn.aggregate([{'$match':{'amenity': 'restaurar
                  {'$group':{'_id':'$opening_hours',
                             'count':{'$sum':1}}},
                  {'$sort':{'count':-1}},
                  {'$limit':10}])

         print(list(restaurant_opening))
```

```
[{'_id': None, 'count': 158}, {'_id': 'Mo-Su 11:00-23:00', 'count': 2}, {'_id': 'Mo
```

```
In [44]: restaurant_opening = paderborn.aggregate([{'$match':{'amenity': 'restaurar
                  {'$group':{'_id':'$cuisine',
                             'count':{'$sum':1}}},
                  {'$sort':{'count':-1}},
                  {'$limit':10}])

         print(list(restaurant_opening))
```

```
[{'_id': None, 'count': 110}, {'_id': 'italian', 'count': 31}, {'_id': 'pizza', 'co
```

1.5 Other ideas about the datasets

1.5.1 Improvement of the current data

When I did some research on OSM and formats or tags used, I stumbled upon some website with local OSM groups and there was also one in Paderborn (which obviously isn't active anymore, but it existed). Furthermore, as shown above, 906 users worked on the data. For that reason I expected to have only a few inconsistencies and problems. That was only partly true.

Phone numbers As mentioned above, the format of the phone numbers was inconsistent. I think I have shown that it is really easy to make them consistent and especially reduce wrongly formatted numbers. I am surprised that nobody did that yet.

Possible improvements Additionally, I think the focus people had when creating the data was not that optimal. It's funny to have detailed data about benches, but I can't think of any practical need for that. In contrast, there are 158 restaurants without any information about opening hours, which is definitely a more important information. It might be difficult to find a good and consistent format, but I think it would be worth implementing and standardizing it.

A suggestion could be: * 2-letter abbreviations for every day: mo, tu, we, th, fr, sa, su * After each abbreviation we use a 4-digit integer representing the opening hours in a 24-hour format * There could be other abbreviations to summarize groups of days (for example "wd" for weekdays) or specify opening hours for holidays. * If the amenity is closed on one day, this could be simply left out.

So if we want to use this schema for a shop with opening hours from 10 to 20 on weekdays, from 10 to 18 on Saturdays and closed on Sundays this could be: "wd1020sa1020"

1.5.2 Further ideas

The first review of this project provided me with a nice idea which I think is quite neat: adding Pokémon Go data to OSM. There are three things which are important for Pokémon Go players: Pokéstops, Gyms and, of course, spawn points of Pokémons. As the latter is changing often it's not possible to represent them on a static map. What is way more interesting are the places of Pokéstops and Gyms. They can be specified by their coordinates - basically the same as any other location.

Benefits: * More users, who could also help to improve the quality of the maps. * Neat tool for all Pokémon Go players as this can be used to plan walking routes (most efficient way to visit x Pokéstops) or find other interesting places.

Anticipated problems: * Pokémon Go data is only interesting for a small number of people compared to the number of people who use OSM. Therefore, it is a large amount of data for a small group of people. * This data adds more temporary data. Although you could say that for other information as well, I think you can say that game data is changing more frequently than data from the “real world”.

```
In [ ]:
```