Wrangling OpenStreetMap Data for South-East Berlin (Neukölln) with Python and MongoDB

In this project, I wrangle OpenStreetMap Data for the South-East of Berlin, with a special focus on the Berlin-Neukölln area in Germany, my home region. In the following sections, I will present how to convert the OSM XML file to JSON to import it into MongoDB. Next, I will inspect the data for inconsistencies and clean up some of the data. Third, I will give some descriptives of the data-set to answer some interesting questions about Saxony-Anhalt.

Obtaining OpenStreetMap Data

I obtained OSM XML data via the overpass API (http://overpass-api.de/query_form.html) for the Neukölln area with the following query `(node(52.3706,13.3154,52.5005,13.6642);<;);out meta;.`

Cleaning the data

First, I need to get an overview of the XML data to be able to later convert it and import it into MongoDB. To get an overview of the tags in the XML, I first iterate over them and list them.

In [66]:
```
import xml.etree.cElementTree as ET
import pprint

infile = 'neukolln.osm'

def count_tags(filename):
    tags = {}
    for events, child in ET.iterparse(filename, events=('start',)):
        if child.tag not in tags:
            tags[child.tag] = 1
        else:
            tags[child.tag] = tags[child.tag] + 1
        child.clear()
    return tags

count_tags(infile)
```

Out[66]:
```
{'member': 104095,
 'meta': 1,
 'nd': 1881691,
 'node': 1557724,
 'note': 1,
 'osm': 1,
 'relation': 4242,
 'tag': 1739250,
 'way': 230341}
```

As the data extract also includes data for Berlin suburbs other than Neukölln, I need to filter the data to exclude those other cities. Relevant data is included in the tag `addr:suburb` tag. The following code shows which other suburbs are included in the data-set. Otherwise, the data for the suburb field appears to be fine.

In [5]:
```
def count_suburbs(filename):
    suburbs = {}
    for events, child in ET.iterparse(filename, events=('start',)):
        if (child.tag == "tag") and (child.attrib['k'] == "addr:suburb"):
            if child.attrib['v'] not in suburbs:
                suburbs[child.attrib['v']] = 1
            else:
                suburbs[child.attrib['v']] = suburbs[child.attrib['v']] + 1
        child.clear()
    return suburbs

count_suburbs(infile)
```

Out[5]:
```
{'Adlershof': 2750,
 'Alt-Mariendorf': 1,
 'Alt-Treptow': 698,
 'Altglienicke': 5900,
 'Baumschulenweg': 2171,
 'Biesdorf': 2756,
 'Bohnsdorf': 4353,
 'Britz': 5566,
 'Buckow': 6354,
 'Charlottenburg': 58,
 'Friedenau': 1635,
 'Friedrichsfelde': 587,
 'Friedrichshagen': 2586,
 'Friedrichshain': 498,
 'Gropiusstadt': 981,
 'Grünau': 1162,
 'Heinersdorf': 1,
 'Johannisthal': 2822,
 'Karlshorst': 3862,
 'Kaulsdorf': 3065,
 'Kreuzberg': 4581,
 'Köpenick': 8536,
 'Lankwitz': 5469,
 'Lichtenberg': 4,
 'Lichtenrade': 10288,
 'Lichterfelde': 4590,
 'Mahlsdorf': 5776,
 'Mariendorf': 6169,
 'Marienfelde': 3701,
 'Müggelheim': 1415,
 'Münchehofe': 8,
 'Neukölln': 6232,
 'Niederschöneweide': 1282,
 'Oberschöneweide': 1544,
 'Plänterwald': 807,
 'Rudow': 10634,
 'Rummelsburg': 527,
 'Schmöckwitz': 1350,
 'Schöneberg': 5949,
 'Schöneweide': 1,
 'Steglitz': 4744,
 'Tempelhof': 5311,
 'Tiergarten': 95,
 'Vorwerk': 3,
 'Waldesruh': 7,
 'Waltersdorf': 8,
 'Waßmannsdorf': 19,
 'Wilmersdorf': 2799}
```

Next, I look for various problems in the data-set to fix them before importing.

Zip-Codes

Next to the suburb information, data on postal codes are another important piece of information for selecting only those nodes that relate to Neukölln. The zip range for Neukölln is 10965 - 12099. The data looks fine.

In [13]:
```
def count_zips(filename):
    zips = {}
    for events, child in ET.iterparse(filename, events=('start',)):
        if (child.tag == "tag") and (child.attrib['k'] == "addr:postcode") and (
int(child.attrib['v']) > 10964) and (int(child.attrib['v']) < 12100):
            if child.attrib['v'] not in zips:
                zips[child.attrib['v']] = 1
            else:
                zips[child.attrib['v']] = zips[child.attrib['v']] + 1
        child.clear()
    return zips

count_zips(infile)
```

Out[13]:
```
{'10965': 1049,
 '10967': 912,
 '10969': 270,
 '10997': 669,
 '10999': 995,
 '12043': 763,
 '12045': 625,
 '12047': 689,
 '12049': 801,
 '12051': 1162,
 '12053': 757,
 '12055': 689,
 '12057': 891,
 '12059': 761,
 '12099': 1326}
```

Street Names

Finally, I check for inconsistencies in naming streets. Street name data is contained in the `addr:street` field. Here, I particularly look for the presence of typical abbreviations for `street` and `place`. There are a few issues that need to be addressed for 3 streets that were abbreviated.

In [49]:
```
def count_streets(filename):
    streets = {}
    for events, child in ET.iterparse(filename, events=('start',)):
        if (child.tag == "tag") and (child.attrib['k'] == "addr:street"):
            if ("str." in child.attrib['v']) or ("pl." in child.attrib['v']):
                if child.attrib['v'] not in streets:
                    streets[child.attrib['v']] = 1
                else:
                    streets[child.attrib['v']] = streets[child.attrib['v']] + 1
        child.clear()
    return streets

count_streets(infile)
```

Out[49]:
```
{'Graefestr.': 1, 'Rheinstr.': 1, 'Weichselstr.': 1}
```

Detecting . in fields later used for MongoDB keys

MongoDB does not accept periods in key fields. Therefore, I need to see if this is a problem in the data-set. The results below illustrate that this is only a minor problem.

In [48]:
```
for events, child in ET.iterparse(infile, events=('start',)):
    if (child.tag == "tag"):
        for tags in child.iter('tag'):
            if '.' in tags.attrib['k']:
                print(tags.attrib)
    child.clear()
```

```
{'k': 'link:berlin.de', 'v': 'http://www.berlin.de/restaurants/2392663-1622830
'}
{'k': 'step.height', 'v': 'high'}
{'k': 'step.condition', 'v': 'even'}
{'k': 'step.height', 'v': 'normal'}
{'k': 'step.length', 'v': 'normal'}
```

Preparing Data-Cleansing Function

There are two issues:

1. cleaning street names
2. replacing "." in the `k` attributes of the `<tag>` elements that will later become keys in MongoDB, which does not allow for `.` in keys.

To achieve this goal, I define a cleaning function.

In [57]:
```
def cleaning_keys(text):
    cleaned = text.replace('.', '_')
    return cleaned

def cleaning_streetname(text):
    cleaned = text.replace('str.', 'straße')
    return cleaned
```

Importing to MongoDB

Next, I import the filtered and cleaned data to a local MongoDB instance. For the import, I will first convert the parsed XML document to JSON format. I only import `node` and `way` information. To save memory usage, I will immediately import the created python dictionary to a local MongoDB instance.

In [67]:
```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017")
db = client.users

def ConvertToJSON(filename):
    #data = [] #could hold the total list of dictionaries, if needed
    for events, child in ET.iterparse(filename, events=('start',)):
        if child.tag == "node" or child.tag == "way":
            entry_dict = {} # will hold the entries for each child
            tag_dict = {} # will hold attributes of <tag> for nodes and ways

            entry_dict = child.attrib
            entry_dict['data_type'] = child.tag
            for tags in child.iter("tag"):
                if 'addr:street' in tags.attrib['k']: # cleaning street names an
d adding to dict
                    tag_dict[cleaning_keys(tags.attrib['k'])] = cleaning_streetn
ame(tags.attrib['v'])
                else: # removing dots from keys and adding to dict
                    tag_dict[cleaning_keys(tags.attrib['k'])] = tags.attrib['v']

            entry_dict.update(tag_dict)
            #data.append(entry_dict)
            db.neukolln.insert_one(entry_dict)
        child.clear()


ConvertToJSON(infile)
```

Analyzing the data

First, here's an overview of the resulting database:

In [69]:
```
# This is a sample entry
pprint.pprint(db.neukolln.find_one({'addr:suburb': 'Neukölln'}))
```

```
{'_id': ObjectId('57b23f583afdf0312caf45b7'),
 'addr:city': 'Berlin',
 'addr:country': 'DE',
 'addr:housenumber': '6',
 'addr:postcode': '12047',
 'addr:street': 'Sonnenallee',
 'addr:suburb': 'Neukölln',
 'changeset': '34571755',
 'data_type': 'node',
 'id': '68363380',
 'lat': '52.4874054',
 'lon': '13.4257380',
 'name': 'Days Inn Berlin City South',
 'phone': '+4930613820',
 'timestamp': '2015-10-11T15:33:17Z',
 'tourism': 'hotel',
 'uid': '2754647',
 'user': 'BenSim78',
 'version': '14',
 'website': 'http://www.euro-hotel.net',
 'wheelchair': 'yes'}
```

In [70]:
```
# The total number of entries
print(db.neukolln.count())
```

```
1788065
```

In [71]:
```
# The total number of entries with an Neukölln suburb entry
print(db.neukolln.count({'addr:suburb': 'Neukölln'}))
```

```
6120
```

In [72]:
```
# The total size of the database
size = db.command({'collstats': 'neukolln'})
print(size['storageSize'])
```

```
121860096
```

In [73]:
```
# Number of nodes
print(db.neukolln.count({'data_type': 'node'}))
```

```
1557724
```

In [74]:
```
# Number of ways
print(db.neukolln.count({'data_type': 'way'}))
```

```
230341
```

Amenities in Neukölln and other suburbs

Neukölln, especially its northern parts, is known for its many restaurants, bars and cafes. Let's look at the data, comparing Neukölln to all other suburbs in the data-set.

In [80]:
```
# count of amenities in Neukölln
query = [
        {'$match': {'addr:suburb': 'Neukölln', 'amenity' : {'$exists' : 'true'}}
},
        {'$group': {'_id': '$amenity', 'count': {'$sum' : 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}
]

for doc in db.neukolln.aggregate(query):
    pprint.pprint(doc)
```

```
{'_id': 'pub', 'count': 89}
{'_id': 'restaurant', 'count': 58}
{'_id': 'cafe', 'count': 45}
{'_id': 'fast_food', 'count': 31}
{'_id': 'kindergarten', 'count': 28}
{'_id': 'bar', 'count': 24}
{'_id': 'post_box', 'count': 22}
{'_id': 'pharmacy', 'count': 17}
{'_id': 'school', 'count': 15}
{'_id': 'place_of_worship', 'count': 14}
```

In [82]:
```
# count of amenities in other south-eastern Berlin areas
query = [
        {'$match': {'addr:suburb': {'$ne': 'Neukölln'}, 'amenity' : {'$exists' :
 'true'}}},
        {'$group': {'_id': '$amenity', 'count': {'$sum' : 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}
]

for doc in db.neukolln.aggregate(query):
    pprint.pprint(doc)
```

```
{'_id': 'bench', 'count': 1828}
{'_id': 'parking', 'count': 1757}
{'_id': 'restaurant', 'count': 1119}
{'_id': 'waste_basket', 'count': 923}
{'_id': 'kindergarten', 'count': 821}
{'_id': 'recycling', 'count': 779}
{'_id': 'post_box', 'count': 689}
{'_id': 'fast_food', 'count': 663}
{'_id': 'cafe', 'count': 596}
{'_id': 'bicycle_parking', 'count': 439}
```

The results show that there is indeed a relatively high presence of restaurants and bars and cafés in Neukölln, compared to its more suburban neighbors.

Next, let's look at the types of cuisines.

In [90]:
```
# count of cuisines in Neukölln
total = db.neukolln.count({'addr:suburb': 'Neukölln', 'cuisine' : {'$exists' : '
true'}})
print("Total entries with cuisine", total)

query = [
        {'$match': {'addr:suburb': 'Neukölln', 'cuisine' : {'$exists' : 'true'}}
},
        {'$group': {'_id': '$cuisine', 'count': {'$sum' : 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}
]

for doc in db.neukolln.aggregate(query):
    pprint.pprint(doc)
```

```
Total entries with cuisine 96
{'_id': 'italian', 'count': 10}
{'_id': 'turkish', 'count': 9}
{'_id': 'asian', 'count': 7}
{'_id': 'coffee_shop', 'count': 7}
{'_id': 'burger', 'count': 6}
{'_id': 'indian', 'count': 6}
{'_id': 'german', 'count': 5}
{'_id': 'pizza', 'count': 5}
{'_id': 'greek', 'count': 3}
{'_id': 'chinese', 'count': 3}
```

In [91]:
```
# count of cuisines in other south-eastern Berlin areas
total = db.neukolln.count({'addr:suburb': {'$ne': 'Neukölln'}, 'cuisine' : {'$ex
ists' : 'true'}})
print("Total entries with cuisine", total)
query = [
        {'$match': {'addr:suburb': {'$ne': 'Neukölln'}, 'cuisine' : {'$exists' :
 'true'}}},
        {'$group': {'_id': '$cuisine', 'count': {'$sum' : 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}
]

for doc in db.neukolln.aggregate(query):
    pprint.pprint(doc)
```

```
Total entries with cuisine 1384
{'_id': 'italian', 'count': 230}
{'_id': 'kebab', 'count': 111}
{'_id': 'german', 'count': 99}
{'_id': 'asian', 'count': 70}
{'_id': 'pizza', 'count': 64}
{'_id': 'burger', 'count': 59}
{'_id': 'indian', 'count': 57}
{'_id': 'turkish', 'count': 56}
{'_id': 'ice_cream', 'count': 49}
{'_id': 'chinese', 'count': 48}
```

So, Italian food is very popular in the whole region. However, german food is more frequent in the suburbs.

Common Names for amenities

Finally, out of curiosity, what's the most common names for restaurants and cafés? There are not that many surprises, here!

In [99]:
```
# Common restaurant names in suburbs
query = [
        {'$match': {'amenity' : {'$eq' : 'restaurant'}, 'name': {'$exists': 'tru
e'}}},
        {'$group': {'_id': '$name', 'count': {'$sum' : 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}
]

for doc in db.neukolln.aggregate(query):
    pprint.pprint(doc)
```

```
{'_id': 'Tomasa', 'count': 3}
{'_id': 'Que Pasa', 'count': 3}
{'_id': 'Sadhu', 'count': 3}
{'_id': 'Restaurant Split', 'count': 3}
{'_id': 'Maria', 'count': 2}
{'_id': 'Don Antonio', 'count': 2}
{'_id': 'Fortuna', 'count': 2}
{'_id': 'Steakhaus Barbecue', 'count': 2}
{'_id': 'Roma', 'count': 2}
{'_id': 'Cancún', 'count': 2}
```

In [101]:
```
# Common cafe names in suburbs
query = [
        {'$match': {'amenity' : {'$eq' : 'cafe'}, 'name': {'$exists': 'true'}}}
,
        {'$group': {'_id': '$name', 'count': {'$sum' : 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}
]

for doc in db.neukolln.aggregate(query):
    pprint.pprint(doc)
```

```
{'_id': 'Tchibo', 'count': 3}
{'_id': 'Tele-Internetcafé', 'count': 2}
{'_id': 'Thoben', 'count': 2}
{'_id': 'Fräulein Frost', 'count': 2}
{'_id': 'Café Obergfell', 'count': 2}
{'_id': 'Brezel Company', 'count': 2}
{'_id': 'Café Melanie', 'count': 2}
{'_id': 'Eis Hennig', 'count': 2}
{'_id': 'Marcello', 'count': 1}
{'_id': 'soup2go', 'count': 1}
```

Contribution of community to the map

How many people have contributed to the map? And who has contributed most? Overall, a large numer of individuals have contributed. However, when looking at the top contributors (e.g. user atpl_pilot (https://www.openstreetmap.org/user/atpl_pilot)) they have contributed tremendously and contributed a large share of the data (about 50% for useratpl_pilot).

In [111]:
```
# User Count
user_count = db.neukolln.distinct('user')
print("Number of unique contributors: ", len(user_count))
```

```
Number of unique contributors:  2152
```

In [112]:
```
# Top Ten Contributors
query = [
        {'$match': {'user' : {'$exists': 'true'}}},
        {'$group': {'_id': '$user', 'count': {'$sum' : 1}}},
        {'$sort': {'count': -1}},
        {'$limit': 10}
]

for doc in db.neukolln.aggregate(query):
    pprint.pprint(doc)
```

```
{'_id': 'atpl_pilot', 'count': 886084}
{'_id': 'jacobbraeutigam', 'count': 183115}
{'_id': 'anbr', 'count': 110080}
{'_id': 'DonRudi', 'count': 62359}
{'_id': 'Balgofil', 'count': 56431}
{'_id': 'g0ldfish', 'count': 50273}
{'_id': 'webpassenger', 'count': 36522}
{'_id': 'geozeisig', 'count': 33281}
{'_id': 'Elwood', 'count': 32581}
{'_id': 'Polarbear', 'count': 22350}
```

Discussion

During the analysis, I encountered the following minor problems while studying the data. Overall, data quality was very good and there were little noticable data-entry errors.

First, the raw data includes a large share of lines of nodes that include information on changes in the data. While this data is relevant and great to have, in a next iteration of this analysis, I would write code to detect the information on the latest time of change and add only this information to the data-base. The change-set information especially distorts information on user contributions.

Second, also, in a future iteration, it would be interesting to check which fields in the data-set, like surface information for streets, or wheelchair-access at buildings are still missing. It would be great to write code that would provide functionality for users to enter their area (lat, lon) and enter a topic that is important to them (like street surface for cyclists, wheelchair-friendly access for handycapped) and then getting a list of the 10 closest nodes that are missing this information.

Implementation could be rather complex when working with the whole data-set. Thus, it would be necessary to limit the extract of the osm data that the user analyzes. For example, one could use a service like the overpass API to get the data. Next, one would need to create a python dictionary holding the information on which tags for specific types of nodes are possible. Then, one would check the osm data for the (non)presence of these tags and return to the user those nodes. The user would then have the possibility to contribute and provide information that is known to him/her. Another larger challenge would be to re-submit the data to the core osm project. The simplest solution from a programming perspective would be to encourage the user to go to the official osm site and add the information there. This would assure data integrity and allow for quality assurance. A solution that is easier for the user, i.e. one that would allow the automatic entry of data into the nodes returned to him/her, would be even more complex. One would have to programmatically push data to the osm core data-set, check data qualtiy, ensure that the data has not been changed since it was last retrieved, etc.

Overall, I enjoyed working with data from the impressive osm data-set. With the tools above, I am now encouraged to use the data for future research. Especially in the social sciences, information about local businesses, densities of public institutions, traffic and other amenities can provide interesting insights into the social structures of cities and their political implications.