Wrangle OpenStreetMap data

Udacity Data Analyst Nanodegree - Project 3

Introduction

Map

- Cologne, http://www.openstreetmap.org/#map=11/50.9387/6.8740 (http://www.openstreetmap.org/#map=11/50.9387/6.8740)
- Before moving to Milan, Italy where I currently live, Cologne has been my home for several years. I know this city quite well and I am curious to improve its OSM data

Resources

- Udacity course materials
- Python 3 documentation (https://docs.python.org/3/)
- MongoDB documentation (https://docs.mongodb.com/manual/)
- MongoDB driver documentation (https://docs.mongodb.com/ecosystem/drivers/python/)
- Markdown documentation (https://daringfireball.net/projects/markdown/syntax)

Generate sample OSM file

In [243]:
```
# file size of original OSM file: 376 MB
!ls -lh cologne_germany.osm
```

```
-rw-r--r--@ 1 stefan  staff   376M Dec 11 17:14 cologne_germany.os
m
```

In [244]:
```
# !/usr/bin/env python
# -*- coding: utf-8 -*-
#
#import xml.etree.ElementTree as ET  # Use cElementTree or lxml if
too slow
#
#OSM_FILE = "cologne_germany.osm"  # Replace this with your osm fil
e
#SAMPLE_FILE = "cologne_germany_sample.osm"
#
#k = 10 # Parameter: take every k-th top level element
#
#def get_element(osm_file, tags=('node', 'way', 'relation')):
#    """Yield element if it is the right type of tag
#
#    Reference:
#    http://stackoverflow.com/questions/3095434/inserting-newlines-
in-xml-file-generated-via-xml-etree-elementtree-in-python
#    """
#    context = iter(ET.iterparse(osm_file, events=('start', 'end'))
)
#    _, root = next(context)
#    for event, elem in context:
#        if event == 'end' and elem.tag in tags:
#            yield elem
#            root.clear()
#
#
#with open(SAMPLE_FILE, 'wb') as output:
#    """
#    Changed output of write to byte objects in order to work with
Python 3.x
#
#    Reference:
#    http://stackoverflow.com/questions/33054527/python-3-5-typeerr
or-a-bytes-like-object-is-required-not-str
#    """
#    output.write(b'<?xml version="1.0" encoding="UTF-8"?>\n')
#    output.write(b'<osm>\n  ')
#
#    # Write every kth top level element
#    for i, element in enumerate(get_element(OSM_FILE)):
#        if i % k == 0:
#            output.write(ET.tostring(element, encoding='utf-8'))
#
#    output.write(b'</osm>')
```

In [245]:
```
# file size of sample OSM file: 38 MB
!ls -lh cologne_germany_sample.osm
```

```
-rw-r--r--  1 stefan  staff    38M Nov 10 22:08 cologne_germany_sa
mple.osm
```

Data exploration

In [246]:
```
# setup environment
import xml.etree.ElementTree as ET
import pprint
import re
```

Helper functions

In [247]:
```
def head_input(input_in, n=20):
    """
    Returns n (default=20) items of a dict/list
    """
    if type(input_in) == type(dict()):
        return dict(list(sorted(input_in.items()))[0:n])
    elif type(input_in) == type(list()):
        return input_in[0:n]
    else:
        return "Non supported input type"
```

Get overview of tags

In [248]:
```
def get_tag_counts(file="cologne_germany_sample.osm"):
    """
    Given a valid XML input file, the function returns a dict of ta
gs and their respective counts
    """
    # create tags dict
    tag_counts = {}

    # open file
    with open(file, "r", encoding="utf8") as f:

        # loop over file
        for event, elem in ET.iterparse(f):

            # check if tag is in dict
            if elem.tag not in tag_counts.keys():

                # if not, add tag as new key
                tag_counts[elem.tag] = 1

            # if so...
            else:

                #increase count of identifed tag
                tag_counts[elem.tag] += 1

    return tag_counts
```

In [249]:
```
# print tag and counts of sample data
get_tag_counts()
```

Out[249]:
```
{'relation': 353, 'way': 30566, 'member': 4594, 'nd': 235134, 'osm
': 1, 'tag': 153299, 'node': 156597}
```

Get overview ot tag keys and values

In [250]:
```
def get_tag_types(file="cologne_germany_sample.osm"):
    """
    Given a valid XML input file, the function returns a dict of keys and corresponding counts of the "tag" attribute
    """
    # create tags dict
    tag_types = {}

    # open file
    with open(file, "r", encoding="utf8") as f:

        # loop over file
        for event, elem in ET.iterparse(f):

            # inspect only "tag" elements
            if elem.tag == "tag":

                # loop over "tag" elements
                for tag in elem.iter("tag"):

                    # check if tag key not in tags_types dict
                    if tag.attrib["k"] not in tag_types.keys():

                        # if not add key with count 1
                        tag_types[tag.attrib["k"]] = 1

                    else:

                        # if so increase count
                        tag_types[tag.attrib["k"]] += 1

    return tag_types
```

In [251]:
```
tag_types = get_tag_types()
```

In [252]:
```
# get first 20 items in tag_types dict
head_input(tag_types)
```

Out[252]:
```
{'TMC:cid_58:tabcd_1:LCLversion': 46, 'TMC:cid_58:tabcd_1:PrevLocationCode': 28, 'FIXME': 39, 'addr:housename': 18, 'addr:country': 13182, 'TMC:cid_58:tabcd_1:Class': 46, 'abandoned': 3, 'TMC:cid_58:tabcd_1:LocationCode': 46, 'TMC:cid_58:tabcd_1:NextLocationCode': 28, 'TMC:cid_58:tabcd_1:TypeName:loc': 1, 'TMC:cid_58:tabcd_1:TypeName': 1, 'addr:district': 1, 'VRS:ortsteil': 196, 'VRS:name': 57, 'VRS:gemeinde': 196, 'VRS:ref': 195, 'access': 712, 'TMC:cid_58:tabcd_1:Direction': 15, 'addr:city': 13392, 'Denkmalnummer': 2}
```

In [272]:
```
# pretty print all items in tag_types dict (produces a long list only used for exploration)
#pprint.pprint(tag_types)
```

In [254]:
```
# print tag_types with more than 200 values
pprint.pprint({(k,v) for k,v in tag_types.items() if v > 200})
```

```
{('access', 712),
 ('addr:city', 13392),
 ('addr:country', 13182),
 ('addr:housenumber', 13337),
 ('addr:postcode', 13351),
 ('addr:street', 13395),
 ('addr:suburb', 246),
 ('amenity', 1414),
 ('barrier', 888),
 ('bicycle', 806),
 ('building', 23639),
 ('building:colour', 243),
 ('building:levels', 5789),
 ('building:part', 353),
 ('building:use', 10941),
 ('created_by', 202),
 ('electrified', 351),
 ('entrance', 837),
 ('foot', 767),
 ('frequency', 334),
 ('gauge', 429),
 ('height', 362),
 ('highway', 5285),
 ('historic', 256),
 ('landuse', 409),
 ('lanes', 596),
 ('layer', 400),
 ('leisure', 231),
 ('lit', 344),
 ('maxspeed', 1014),
 ('name', 3704),
 ('natural', 468),
 ('note', 298),
 ('oneway', 938),
 ('operator', 702),
 ('public_transport', 280),
 ('railway', 1124),
 ('ref', 836),
 ('roof:colour', 288),
 ('roof:orientation', 437),
 ('roof:shape', 1325),
 ('service', 755),
 ('shop', 453),
 ('source', 3033),
 ('surface', 770),
 ('type', 357),
 ('voltage', 335),
 ('website', 459),
 ('wheelchair', 562)}
```

Explore "fixme" and "fixed" tag keys

In [255]:
```
def get_tag_key(file="cologne_germany_sample.osm", key="FIXME"):
    """
    Given a valid XML input file, the function returns a list of va
lues for the corresponding key of the "tag" attribute
    """
    # create tags dict
    tag_keys = []

    # open file
    with open(file, "r", encoding="utf8") as f:

        # loop over file
        for event, elem in ET.iterparse(f):

            # inspect only "tag" elements
            if elem.tag == "tag":

                # loop over "tag" elements
                for tag in elem.iter("tag"):

                    # check if tag key not in tags_types dict
                    if tag.attrib["k"] == key:

                        # if not add key with count 1
                        tag_keys.append(tag.attrib["v"])

                    else:
                        continue

    return set(tag_keys)
```

In [256]:
```
tag_key_fixme = get_tag_key()
```

Most of the fixme notes are related to errors concerning buildings, e.g. where an entrance is located

In [259]:
```
# pretty print FIXME notes
pprint.pprint(tag_key_fixme)
```

```
{'Bitte Details ergänzen',
 'Bitte Existenz des Defi prüfen.',
 'Bitte Gebäude, Gebäudeteil oder Eingang zuordnen',
 'Bitte näher bezeichnen. barrier=fence?',
 'Diese Landuse Relation sollte man verkleinern',
 'Gebäudeumrisse prüfen',
 'Verbindung?',
 'auch Eingang Ehrenstraße 2',
 'bessere Beschreibung erforderlich',
 'bitte Gebäude oder Gebäudeteil zuordnen',
 'bitte Gebäude zuordnen',
 'bitte Gebäude zuordnen (auf Bild in 12/2013 nicht vorhanden)',
 'bitte Gebäude, Gebäudeteil oder Eingang zuordnen',
 'bitte Gebäudeeingang oder Gebäudeteil zuordnen',
 'bitte Gebäudeeingang zuordnen',
 'bitte Gebäudeingang oder Gebäudeteil zuordnen',
 'bitte Hauseingang bzw. Gebäudeteil zuordnen',
 'bitte Hauseingang oder Gebäudeteil zuordnen',
 'bitte Name und Details ergänzen',
 'bitte Nutzungsart ergänzen',
 'bitte genau zuordnen',
 'bitte richtig zuordnen',
 'ist dieser Abschnitt Einbahnstraße?',
 'lage geschätzt, Juni 2013',
 'landuse=grass für diese großen Planzkästen scheint mir etwas ove
rdressed',
 'maxspeed prüfen: unterschiedlich nach Richtung? Wo Grenze zwisch
en 50 und '
 '60?',
 'route_master sind für PTv2-Member',
 'where does this oneway go?',
 'wirklich residential und nicht path/track?'}
```

In [260]:
```
tag_key_fixed = get_tag_key(key="fixed")
```

Most of the fixed tages are related to roads and how they are categorized

In [261]:
```
# pretty print FIXED tags
pprint.pprint(tag_key_fixed)
```

```
{'Amsterdamer Straße ist stadteinwärts die Fortsetzung der Industr
iestr. '
 '(Trunk)  und liegt auf der optimalen Route für den Fernverkehr a
us der '
 'Innenstadt in Richtung Ruhrgebiet, Hannover, Bremen, Berlin und
Hamburg. '
 'Deshalb als primary road kennzeichnen!',
 'Amsterdamer Straße ist stadteinwärts die Fortsetzung der Industr
```

```
iestr. '
 '(Trunk) und liegt auf der optimalen Route für den Fernverkehr au
s der '
 'Innenstadt in Richtung Ruhrgebiet, Hannover, Bremen, Berlin und
Hamburg. '
 'Deshalb als primary road kennzeichnen!',
 'Auf B 59 stadteinwärts wird Verkehr Ri. Zentrum nach links auf d
ie Äußere '
 'Kanalstr. gewiesen (wegen Verkehrsberuhigung) To Do: Steht der W
egweiser '
 'noch? Wenn ja, für fixme-Abschnitte highway=secondary setzen, da
keine '
 'Fern- u. Regionalnetz-Funktion mehr!',
 'Auf B 59 stadteinwärts wird Verkehr Ri. Zentrum nach links auf d
ie Äußere '
 'Kanalstr. gewiesen. To Do: Steht der Wegweiser noch? Wenn ja, Su
bbelrather '
 'Str. als Parallele zur Venloer Str. (verkehrsberuhigt) regionale
'
 'Netzfunktion. Deshalb highway=secondary',
 'Einseitig isolierte Kennzeichnung als "tertiary road" am S-Bahnh
of '
 'K-Dellbrück. Vorschlag: Kennzeichnung durch die Bahnunterführung
und auf '
 'der Diepeschrather Str. fortsetzen. Alternative: als "residental
road" '
 'zurückstufen.',
 'Kennzeichnung der gesamten Poststraße als "tertiary road" um die
einseitig '
 'isolierte Kennzeichnung als "tertiary road" an der Unterführung
'
 'Bahnstrecke Köln-Troisdorf aufzuheben. Ausbauzustand und Kennzei
chnung '
 '(Verkehrsschilder und Fahrbahnmarkierung) erl',
 'Liegt auf einer leistungsfähigeren Umfahrung von Karolingerring,
'
 'Chlodwigplatz und Ubierring (Rückbau, verkehrsberuhigt) in Richt
ung '
 'Verteilerkreis Süd, was auch durch eine DTV > 20.000 belegt ist.
To Do: '
 'Survey, ob etwas gegen highway=primary spricht.',
 'Liegt auf einer leistungsfähigeren Umfahrung von Karolingerring,
'
 'Chlodwigplatz und Ubierring (Rückbau, verkehrsberuhigt) in Richt
ung '
 'Verteilerkreis Süd. Survey ergab: Route über Bischofsweg, Markts
tr., '
 'Bonner Str. eher primary',
 'Survey by Google Street View: Rochusstraße zwischen Magaretastra
ße und '
 'Venloer Str. ist keine Vorfahrtstraße, die Geschwindigkeit auf 3
0 km/h '
 'begrenzt, teilweise sehr eng und Einbahnstraße. To Do: Survey by
car. '
```

```
 'Erwartetes Ergebnis: residental road.'}
```

Explore other interresting tags

The correct key to indicate districts seems to be "addr:suburb" since "addr:district" yields only district of Cologne

In [262]:
```
get_tag_key(key="addr:district")
```

Out[262]:
```
{'Porz'}
```

In [22]:
```
get_tag_key(key="addr:suburb")
```

Out[22]:
```
{'Brück',
 'Buchforst',
 'Deutz',
 'Grengel',
 'Kalk',
 'Lindenthal',
 'Mülheim',
 'Nippes',
 'Ostheim',
 'Poll',
 'Sielsdorf',
 'Urbach',
 'Vingst'}
```

I was wondering which information was stored with "addr:housename". Apparently no standard has been definied, since it is used for clubs, companies, opening times, addresses, etc - an opportunity for improvement? :-)

In [23]:
```
get_tag_key(key="addr:housename")
```

Out[23]:
```
{'Bayburt Kulturverein',
 'Bürgerhaus Stollwerck',
 'C103',
 'Caritas Hospiz Johannes-Nepomuk-Haus',
 'Doc-PT Praxis für Innere- und Allgemeinmedizin',
 'Erik Wickberg-Haus',
 'Feilenhof',
 'HERZBERGMEDIA',
 'Jüdisches Wohlfahrtszentrum',
 'Kartonagenfabrik Seybold',
 'Mo-Fr 09:30-18:30; Sa 09:30-16:00',
 'Post Office',
 'Raderthalgürtel',
 'Schaltwerk',
 'SkinWorks',
 'TrauerHaus Müschenborn',
 'Vereinsheim ESV Olympia Köln e.V.',
 'Villa Hahnenburg'}
```

Sanity check passed, all documents containing "addr:country" refers to Germany (DE = Deutschland = Germany)

In [62]:
```
get_tag_key(key="addr:country")
```

Out[62]:
```
{'DE'}
```

The tag "add:city" yields another opportunity for improvement. Strictly speaking, the only valid value is "Köln" (= Cologne in German). All other values are related to either districts of Cologne or different cities

In [63]:
```
get_tag_key(key="addr:city")
```

Out[63]:
```
{'Bergisch Gladbach', 'Hürth', 'Köln', 'Köln Rath/Heumar', 'Köln-N
ippes'}
```

I was curious about the quality of street names and decided to do an upfront visual check. Suprisingly the quality of street names in Cologne's OSM data is very good: An obvious case for trouble is the spelling of street names in Germany (Strasse vs. Straße, vs Str.) which does not seem to be an issue at all here (Note that Straße and straße is correct, since we do distinguish between street with identical names this way in Germany.)

In [64]:
```
tag_key_street = get_tag_key(key="addr:street")
```

In [279]:
```
# pretty print tag_key_street (produces a long list only used for e
xploration)
#pprint.pprint(tag_key_street)
```

Unfortunately Cologne's data contains two tags for storing postal codes, "addr:postcode" and "addr:postal_code". Both should be merged into a single tag

In [65]:
```
get_tag_key(key="addr:postcode")
```

```
Out[65]: {'50354',
          '50667',
          '50668',
          '50670',
          '50672',
          '50674',
          '50676',
          '50677',
          '50678',
          '50679',
          '50733',
          '50735',
          '50737',
          '50739',
          '50765',
          '50767',
          '50823',
          '50825',
          '50827',
          '50829',
          '50858',
          '50859',
          '50931',
          '50933',
          '50935',
          '50937',
          '50939',
          '50968',
          '50969',
          '50996',
          '50997',
          '50999',
          '51061',
          '51063',
          '51065',
          '51067',
          '51069',
          '51103',
          '51105',
          '51107',
          '51109',
          '51143',
          '51145',
          '51147',
          '51149',
          '51427',
          '51467',
          '51469'}
```

In [68]:
```
get_tag_key(key="postal_code")
```

Out[68]:
```
{'50668',
 '50672',
 '50674',
 '50733',
 '50765',
 '50767',
 '50935',
 '50937',
 '50996',
 '51061',
 '51063',
 '51065',
 '51067',
 '51069',
 '51103',
 '51109',
 '51143,51145',
 '51145',
 '51147'}
```

Opening hours are mess lacking any standard

In [273]:
```
get_tag_key(key="opening_hours")
```

Out[273]:
```
{'20:00-02:00', 'Mo-Fr 08:30-20:00, Sa 08:00-20:00', 'Mo 12:00-19:
00; Tu-Fr 10:00-20:00; Sa 09:00-16:00', 'Mo-Fr 09:00-18:00; Sa, Su
, PH off', 'Mo-Fr 10:00-13:00; Mo-Fr 14:00-18:00; Sa 10:00-13:00',
'Mo-Sa 12:00-18:00', 'Mo-Sa 06:00-23:00', 'Mo-Fr 08:30-18:30; Sa 0
9:00-13:00', 'Mo,Tu 08:30-18:30; We 08:30-13:30, 14:30-18:30; We 0
8:30-18:30; Fr 08:30-13:30, 14:30-18:30; Sa 09:00-13:00', 'Mo-Th 0
9:00-12:00,14:00-17:00; Fr 09:00-12:00', 'Mo-Fr 08:30-19:00; Sa 08
:30-18:30; Su 10:30-18:00', 'Mo-Fr 11:00-19:00; Sa 10:00-16:00; Su
, PH off', 'Mo-Th 08:00-19:00; Fr 08:00-18:00; Sa 09:00-14:00', 'M
o-Th 15:00-01:00, Fr 15:00-02:00, Sa 12:00-02:00, Su,PH 12:00-01:0
0', 'Mo-Sa 11:30-19:00', 'Mo-Fr 09:30-18:00', 'Su-Th 11:00-23:00;
Fr,Sa 11:00-02:00', 'Mo-Tu, Th-Fr 10:00-13:00, 15:00-18:30; We, Sa
10:00-13:00', 'Mo-Fr 09:00-18:30; Sa 10:00-14:00', 'Mo-We 06:00-23
:00; Fr 06:00-24:00; Sa 07:00-24:00; Su 08:00-23:00', 'Mo,Tu 08:30
-13:30, 14:30-18:30; We 08:30-14:00; Th,Fr 08:30-13:30,14:30-18:30
; Sa 08:30-14:00', 'Mo-Fr 08:30-19:00; Sa 09:00-14:00', 'Mo, Th, F
r 12:00-20:00; Tu, We 15:00-20:00; Sa 10:00-20:00; Su, PH off', 'M
o-Fr 08:30-19:00; Sa 09:30-16:00', 'Mo-Fr 11:00-23:00; Sa 11:00-22
:00', 'Mo-So 22:00 - 05:00, Sa-So ab 06:00', 'Mo-Fr 12:00+; Sa 18:
00+', 'Mo-Sa 10:00-20:00', 'Mo-Fr 08:00-20:00; Sa 09:00-20:00', 'M
o-Su, PH 12:00-23:30', 'Feb-Nov: 09:30-17:45', 'Mo-Th 18:00-00:00;
Fr, Sa 18:00-03:00', 'Mo-Fr 11:00-18:30', 'Mo-Su 09:00+', 'Mo-Fr 0
8:00-19:00;Sa 10:00-14:00', 'Mo-Sa 08:00-21:00', 'Mo-Sa 15:00-20:0
0', 'Mo-Sa 09:00+; Su 11:00+', 'Mo-Th 13:00-18:30; Fr 10:00-18:30;
Sa 10:00-14:00', 'Mo-Sa 08:00-20:00; Su, PH off', 'Mo-Fr 08:30-19:
```

00; Sa 09:00-16:00', 'We 15:00-17:00; Su 10:00-12:00', 'Mo-Su 11:0
0-15:00, 17:00-24:00', 'Mo-Sa 10:30-23:00; Su 09:00-18:00', 'Mo-Fr
10:00-19:00; Sa 10:00-18:00', 'Tu-Su 12:30-22:00', 'Mo-Su 17:00-01
:00', 'Tu-Th 17:00-24:00, Fr-Sa 17:00-01:00, Su 17:00-24:00', 'Mo-
Sa 07:00-22:00; Su, PH off', 'Mo-Sa 09:00-19:00', 'Mo-Fr 09:30-19:
00; Sa 09:30-18:00', '10:00-24:00', 'Mo-Sa 18:00-00:00; Su off', '
Mo-Fr 08:30-17:00', 'Mo-Fr 10:00-19:00; Sa 10:00-18:00; Su, PH off
', 'Mo off; Tu-Th 20:00-01:00, Fr-Sa 20:00-03:00, Su 19:00-01:00',
'Mo-Fr 09:00-13:00, 14:00-18:00', 'Mo-Fr 08:00-18:30; Sa 08:00-13:
00', 'Mo-Sa 07:00-21:00; Su, PH off', 'Mo off; Tu-Su 10:00-17:00',
'Mo-Fr 10:00-19:00;Sa 10:00-17:00', 'Mo-Fr 08:00-18:30; Sa 07:30-1
5:00, Sa 07:30-18:00 "im Außenbereich"; Su 11:00-13:00', 'Mo,Tu 08
:30-13:00,14:30-18:30;We 08:30-13:00;Th,Fr 08:30-13:00,14:30-18:30
;Sa 09:00-13:00', 'Mo-Fr 08:00-18:30; Sa 08:30-13:30', 'Tu-Sa 12:0
0-15:00; 17:00+; Su 12:00+', 'Mo,Tu 08:00-13:00,14:00-18:30; We 08
:00-13:00; Th,Fr 08:00-13:00,14:00-18:30;Sa 08:30-13:00', 'Mo-Th 1
8:00-01:00, Fr-Sa 18:00-03:00', 'Mo-Fr 06:30-18:30; Sa 06:00-13:00
; Su 08:00-12:00', 'Mo-Fr 09:00-19:00; Sa 09:00-18:00; Su, PH off'
, 'We, Fr 08:00-13:00; Mo, Tu, Th 08:00-13:00, 14:30-18:00', '09:3
0-17:45', 'Tu-Fr 10:00-19:00; Sa 10:00-17:00; Su, PH off', 'Mo 09:
00-12:00; We 15:00-17:00', 'Mo-Fr 08:00-17:00', 'Mo-Fr 09:30-19:00
; Sa 09:30-16:00', 'Tu 10:00-12:00; We 16:00-18:00; Su 10:00-12:00
', 'Mo-Fr 08:30-13:00,15:00-18:30; Sa 08:30-13:00', 'Mo-Fr 10:30 -
18:30, Sa 10:30 - 16:00', 'Mo-Su 11:30-23:30', 'Mo-Fr 08:30-18:30;
Sa 08:30-13:00', 'Mo-Fr 10:00-13:00,14:00-18:00; Sa 10:00-14:00',
'Oct-Mar 10:00-16:00; Apr-Sep 10:00-18:00', 'Mo-Sa 11:00-19:30', '
Mo, Tu, Th, Fr 09:00-13:00, 14:00-18:00; Sa 10:00-13:00; We, Su, P
H off', 'Mo-Fr 10:00-13:00, 14:00-18:00; Sa 10:30-13:00; Su, PH of
f', 'Mo', 'Mo-Fr 09:00-19:00; Sa 09:00-18:00', 'Mo-Fr 11:30-24:00;
Sa 00:00-01:00,11:30-24:00; Su 00:00-01:00,11:30-23:00', '6:00-23:
00', 'Mo-Sa 07:00-19:00', '24/7', 'Mo-Sa 10:00-20:00; Su, PH off',
'05:00-24:00', 'Mo-Su 12:00-01:00', 'Mo-Fr 10:00-20:00; Sa 10:00-1
9:00; Su, PH off', 'Mo-Sa 10:00-19:00', 'Tu-Fr 12:00-18:30; Sa 12:
00-16:30; Su,Mo,PH off', 'Mo-Fr 09:00-19:00; Sa 10:00-16:00; Su, P
H off', 'Mo-Fr 09:00-20:00;Sa 09:00-16:00', 'Mo-Fr 09:00-19:00; Sa
09:00-17:00', 'Mo-Fr 8:00-18:00; Sa 8:00-12:00', 'Mo off; Tu-Su, P
H 11:00-17:00; Dec24, Dec 25, Dec 31, Jan 01 off || "Jeden ersten
Donnerstag im Monat – KölnTag (außer an Feiertagen) 11 - 22 geöffn
et; Karneval (Weiberfastnacht bis Karnevalsdienstag) geschlossen"'
, 'Mo-Th 12:00-22:00; Fr-Su 12:00-22:30', 'Mo-Fr 07:45-18:00; Sa,
Su, PH off', 'Mo-Sa 16:00-24:00; Su 10:00-24:00', 'Mo,Tu 09:00-13:
00, 15:00-18:30; We 09:00-13:00; Th,Fr 09:00-13:00, 15:00-18:30; S
a 09:00-13:00', 'Mo-Sa 11:00-14:30,16:00-24:00; Tu off; Su 10:30-1
4:30,16:30-24:00', 'Mo-Th, Su 11:00-03:00; Fr-Sa 11:00-05:00', 'Mo
-Fr 08:30-18:30; Sa 09:00-14:00', 'Mo-Fr 06:00-22:00, Sa 07:00-22:
00; Su 08:00-22:00', 'Tu-Fr 12:00-15:00,17:00-23:00; Sa-Su 12:00-2
3:00; Mo off', 'Fr-Sa 23:00+', 'Mo-Fr 11:00-19:00; Sa 11:00-18:00;
Su, PH off', 'Mo-Su 11:00-01:00', 'Mo-Th 10:00-14:00', '11:30-00:3
0', 'Mo-Fr 11:00-19:00; Sa 10:00-18:00; Su, PH off', 'Mo-Th 07:00-
15:30; Fr 07:00-15:00; PH off', 'Mo-Sa 11:00-23:00; Su 14:30-22:30
', 'Mo-Fr 12:00-14:00, 18:00-24:00; Sa 18:00-24:00; Su 12:00-14:00
, 18:00-24:00', 'Mo-Th,Su off; Fr,Sa 21:00-04:00', 'Mo-Fr 08:00-18
:30, Sa 08:30-14:00', 'Fr-Sa 20:00+', 'Mo-Sa 11:00-13:00|| Mo-Fr 0
9:00-19:00 "de:nach Vereinbarung; en:on appointment"', 'Mo-Fr 09:0

```
0-18:30; Sa 09:00-13:00', 'Su -Fr 11:45-14:30, 18:00-23:30; Sa 17:
30-23:30', 'Mo-Fr 09:00-19:00; Sa-Su 10:00-19:00', 'Mo-Sa 10:00-23
:00; Su,PH 10:00-22:00; Dec 24 10:00-15:00; Dec 31 10:00-06:00, Ja
n 1 12:00-22:00', 'Mo-Su 11:00-03:00', 'Mo-Th 08:00-12:00,14:00-18
:00; We,Fr 08:00-12:00', 'Mo off; Tu-Sa 18:30-22:00; Su off', 'Mo
06:00-13:00,14:00-17:00; Tu-We 08:00-13:00; Th 08:00-13:00,14:00-1
9:00; Fr 08:00-13:00', 'Mo,Tu 08:30-13:00, 15:00-18:30; We 08:30-1
3:00; Th,Fr 08:30-13:00,15:00-18:30; Sa 08:30-13:00', 'Mo-Su 11:30
-24:00', 'Mo-Th 09:00-21:00; Fr,Sa 09:00-22:00', 'Mo-Fr 06:00-18:3
0; Sa 06:00-13:00; Su 08:00-16:00', 'Mo-Fr 06:45-17:00', 'Mo-Fr 10
:00-19:00; Sa 10:00-16:00', 'Mo-Su 08:00-23:00', 'Mo-Fr 05:00-18:0
0;Sa 06:00-13:00', 'Mo-Fr 08:00-18:30; Sa 07:30-14:00', 'Mo-Th 09:
00-13:00,14:00-18:00; We 09:00-13:00,14:00-17:30; Fr 09:30-13:30',
'Mo-Sa 11:00-20:00', 'Mo,Tu 08:30-13:30, 14:30-18:30; We 08:30-13:
30, 14:30-18:00; Th,Fr 08:30-13:30, 14:30-18:30; Sa 09:00-13:00',
'Mo,Tu 08:30-13:00, 15:00-18:30; We 08:30-13:0; Th 08:30-13:00, 15
:00-18:30; Fr 08:30-13:30, 14:30-18:30; Sa 08:30-13:00', 'Mo-Sa 09
:00+; Su 14:00+', 'Mo-Fr 08:30-19:00; Sa 09:00-18:00', 'Mo-Sa 07:0
0-22:00', 'Mo,We-Su,PH 12:00-17:00; Tu off', 'Tu-We 10:00-19:00; T
h 11:00-21:00; Fr 10:00-19:00; Sa 09:00-16:00', 'Tu-Su 11:30-22:00
', 'Mo-Sa 09:00-20:00', 'Mo-Fr 10:00-19:00', 'Mo-Fr 09:00-19:00; S
a 09:00-14:30', 'Tu-Sa 12:00-19:00', 'Mo,Tu,Th, Fr 08:00-19:00; Sa
, Su 10:00-19:00', 'open; We, Sa 06:00-14:00 off', 'Mo-Sa 07:00-24
:00', 'Mo-Fr 09:00-18:30; Sa 09:00-14:00', 'Mo-Fr 09:30-23:30; Sa-
Su 10:00-22:00', 'Mo-We 08:30-13:00,14:00-16:30; Th 08:30-13:00,14
:00-18:30; Fr 08:30-13:00', 'Mo-Fr 11:00-23:00; Sa 10:00-23:00', '
Mo-Sa 07:00-21:00', 'Mo-Fr 07:00-24:00; Sa 07:00-22:00', 'Su-Th 11
:30-23:00; Fr-Sa 11:30-23:30'}
```

Lastly, I was curious about the meaning of "alt_name" and "information". Whereas "alt_name" seems to store names for buildings, information seems to store some data related to hiking

In [66]:
```
get_tag_key(key="alt_name")
```

Out[66]:
```
{'Chaussée Brunehaut (BE, F)',
 'Deutz-Thermalbad',
 'Dorint Sofitel An der Messe',
 'Entschlafen der Gottesmutter',
 'Gymnasium - Kaiserin-Theophanu-Schule',
 'Haus der Begegnung',
 'Kölner Seilbahn',
 'LVR-Klinik Forensische Psychiatrie',
 'Rhine Route - part Germany',
 'Stotzheim'}
```

In [67]:
```
get_tag_key(key="information")
```

Out[67]:
```
{'board', 'guidepost', 'hikingmap', 'map', 'nature'}
```

Audit tag keys

Using regular expressions definied during the courss the data is audited

In [263]:
```
# complie regular expressions
lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n\-]')
```

In [264]:
```
def get_audit_tags(file="cologne_germany_sample.osm"):
    """
    tbd
    """
    # create tags dict
    audit_tags = {"lower": 0, "lower_colon": 0, "problemchars": 0,
"other": 0}
    problemchars_list = []

    # open file
    with open(file, "r", encoding="utf8") as f:

        # loop over file
        for _ , elem in ET.iterparse(f):

            if elem.tag == "tag":

                # loop over tags of element
                for tag in elem.iter("tag"):

                    # check for lower
                    if re.search(lower, tag.attrib["k"]):

                        # increase count
                        audit_tags["lower"] += 1

                    # check for lower_colon
                    elif re.search(lower_colon, tag.attrib["k"]):

                        # increase count
                        audit_tags["lower_colon"] += 1

                    # check for problemchars
                    elif re.search(problemchars, tag.attrib["k"]):

                        # add value to problemchars list
                        problemchars_list.append(tag.attrib["k"])

                        # increase count
                        audit_tags["problemchars"] += 1


                # else assign other
                else:
                    audit_tags["other"] += 1

    return audit_tags, set(problemchars_list)
```

In [265]:
```
problemchars_count, problemchars = get_audit_tags()
```

In [268]:
```
# print problemchar count per category
problemchars_count
```

Out[268]:
```
{'other': 153299, 'lower': 61444, 'problemchars': 63, 'lower_colon
': 89518}
```

In [269]:
```
# print problemchars
problemchars
```

Out[269]:
```
{'step.height', 'step.length', 'step.condition', 'surface.material
', 'strassen-nrw:abs'}
```

Problems encountered

A brief summary of the problems encountered during data exploration:

- The city key does not only store the value "Köln" (Cologne in German), but information regarding districts and cities other than Cologne
- Addresses are supringsingly well formated and are *not* considered an issue
- Two keys are used to store information regarding Cologne' districts (addr:suburb and addr:district) and postal codes (postal_code vs postcode)
- Opening hours lack a standard schema for notation
- Analyis of problemchars gave a good overview of potentially intresting data points
- Interestingly, OSM users themselves repoted data point to be fixed using the FIXME tag

Fixing problems

The following function is a prototype for mixing issues with OSM data. It accepts fixes via predefinied mappings and can make use of problemchars identified by the get_audit_tags() function.

In [126]:
```
mapping = {"Köln Rath/Heumar" : "Köln",
           "Köln-Nippes": "Köln",
           "51143,51145" : "51143"}
```

In [172]:
```
def fix_problems(file="cologne_germany_sample.osm", mapping=mapping
, problemchars=problemchars):
    """
    Given a valid XML input file, the function applies fixes to key
s of the "tag" attribute given a mapping dict
    and a list of problemchars
    """
    fixed_dict = {}
    tag = None
    issue = None
```

```
    with open(file, "r", encoding="utf8") as f:

        # loop over file
        for event , elem in ET.iterparse(f):

            # only process "tag" tags
            if elem.tag == "tag":

                # loop over tags of element
                for tag in elem.iter("tag"):

                    # check if value is in mapping dict
                    if tag.attrib["v"] in mapping.keys():

                        # record problematic key
                        issue = tag.attrib["v"]

                        # loop over key value pairs in mapping dict
                        for m in mapping:

                            # apply fixes
                            tag.attrib["v"] = tag.attrib["v"].repla
ce(m, mapping[m])

                        # record fix
                        fix = tag.attrib["v"]

                    # check if key is inproblemchars list
                    elif tag.attrib["k"] in problemchars:

                        # record problematic key
                        issue = tag.attrib["k"]

                        # check for hiphen
                        if re.search(re.compile(r'\-'), tag.attrib[
"k"]):

                            # fix hiphen
                            tag.attrib["k"] = tag.attrib["k"].repla
ce("-", "_")

                        # check for dot
                        elif re.search(re.compile(r'\.'), tag.attri
b["k"]):

                            # fix dot
                            tag.attrib["k"] = tag.attrib["k"].repla
ce(".", ":")

                        # record fixed key
                        fix = tag.attrib["k"]

                    # update fixed_dict
                    if issue and fix:
```

```
                            fixed_dict[issue] = fix

    return fixed_dict
```

In [173]:
```
test = fix_problems()
test
```

Out[173]:
```
{'51143,51145': '51143',
 'Köln Rath/Heumar': 'Köln',
 'Köln-Nippes': 'Köln',
 'step.condition': 'step:condition',
 'step.height': 'step:height',
 'step.length': 'step:length',
 'strassen-nrw:abs': 'strassen_nrw:abs',
 'surface.material': 'surface:material'}
```

Data overview

Prepare XML data for ingest into MongoDB

In [239]:
```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.cElementTree as ET
import pprint
import re
import codecs
import json

CREATED = ["version", "changeset", "timestamp", "user", "uid"]

def shape_element(element):

    # initialize node dict
    node = {}

    # initialize helper dicts and lists
    created = {}
    address = {}
    pos = []
    node_refs = []

    # only process "node" or "way" tags
    if element.tag == "node" or element.tag == "way":

        # add note type to node dict
        node["type"] = element.tag

        # loop over keys of element attributes
        for key in element.attrib:
```

```
            # check if attribute is id or visible
            if key == "id" or key == "visible":

                # add key and value to node dict
                node[key] = element.attrib[key]

            # check if key in created array
            elif key in CREATED:

                # add key and value to created dict
                created[key] = element.attrib[key]

            # check if attribute is lat
            elif key == "lat":

                # cast attribute to float and add to pos list
                pos.insert(0,float(element.attrib[key]))

            # check if attribute is lon
            elif key == "lon":

                # cast attribute to float and add to pos list
                pos.insert(1,float(element.attrib[key]))

            # if not
            else:

                # ignore
                continue

        # loop over children of element
        for child in element:

            # check for node references
            if child.tag == "nd":

                # add ref attribute to node_refs list
                node_refs.append(child.attrib["ref"])

            else:

                # check if child key contains problematic character
s
                if re.search(problemchars, child.attrib["k"]):

                    # ignore
                    continue

                # check if child key does not contain more than 1 c
olon:
                elif not child.attrib["k"].count(":") > 1:

                    # check if attribute key is in mapping
```

```
                    if child.attrib["k"] in mapping.keys() or child
.attrib["v"] in mapping.keys():

                        # loop over mapping dict
                        for m in mapping:

                            # update key according to mapping
                            child.attrib["k"] = child.attrib["k"].r
eplace(m, mapping[m])

                            # update value according to mapping
                            child.attrib["v"] = child.attrib["v"].r
eplace(m, mapping[m])

                    # check if attribute key starts with "addr:"
                    if child.attrib["k"].startswith("addr:"):

                        # clean key
                        clean_key = child.attrib["k"].replace("addr
:", "")

                        # add key and value to address dict
                        address[clean_key] = child.attrib["v"]

                    # if not
                    else:

                        # add key and value to other dict
                        node[child.attrib["k"]] = child.attrib["v"]

                # if it does contain more than 1 colon:
                else:

                    # ignore
                    continue

        # add helper dicts and list to node dict
        if created:

            # add k,v for created dict
            node["created"] = created

        # if pos list contains elements
        if pos:

            # add k,v for pos list
            node["pos"] = pos

        # if address dict contains elements
        if address:

            # add k,v for address dict
            node["address"] = address
```

```
        # if node_refs dict contains elements
        if node_refs:

            # add k,v for node_refs dict
            node["node_refs"] = node_refs

        return node

    else:

        return None
```

In [240]:
```
# code from case study

def process_map(file_in, pretty = False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return data
```

In [241]:
```
data = process_map("cologne_germany_sample.osm", pretty=False)
```

In [223]:
```
# check if data has been processed correctly
data[-1]
```

Out[223]:
```
{'created': {'changeset': '43410968',
  'timestamp': '2016-11-04T21:58:30Z',
  'uid': '16478',
  'user': 'Raymond',
  'version': '1'},
 'highway': 'footway',
 'id': '451465982',
 'node_refs': ['4482995966', '2141100739'],
 'type': 'way'}
```

Inserting data into MongoDB

Using mongoimport from UNIX shell in virtual machine:

```
mongoimport --db osm --collection cologne --type json --file /vagrant/c
ologne_germany_sample.json
```

In [12]:
```
# file size of sample JSON: 41 MB
!ls -lh cologne_germany_sample.json
```

```
-rw-r--r--  1 stefan  staff    41M Nov 13 14:46 cologne_germany_sa
mple.json
```

Querying data from MongoDB

In [1]:
```
# setup pymongo, connect to MongoDB and select osm database
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017")
db = client.osm.cologne
```

In [50]:
```
# test connection
db.find_one()
```

Out[50]:
```
{'_id': ObjectId('58287fc8a300d8a50baf9881'),
 'created': {'changeset': '9176870',
  'timestamp': '2011-08-31T14:41:21Z',
  'uid': '81244',
  'user': 'rurseekatze',
  'version': '3'},
 'id': '160058',
 'pos': [50.8964366, 6.9105618],
 'type': 'node'}
```

In [129]:
```
# helper function to display cursor
def get_cursor(query):
    """
    Given a valid MongoDB aggregation query, the function returns i
ts result as a list
    """
    print(list(query))
```

Summary statistics

In [5]:
```
# number of documents in database
db.find().count()
```

Out[5]:
```
187163
```

In [51]:
```
# number of nodes
db.find({"type": "node"}).count()
```

Out[51]: 156596

In [52]:
```
# number of ways
db.find({"type": "way"}).count()
```

Out[52]: 30563

In [61]:
```
# number of unique users
len(db.distinct("created.user"))
```

Out[61]: 886

In [185]:
```
# top 5 users by number of created documents
user_query = db.aggregate([
        {"$group": {"_id": "$created.user", "count": {"$sum": 1}}},
        {"$sort": {"count": -1}},
        {"$limit" : 5}
    ])
```

The top 5 users created 70% of the documents in the database (130536/187163)

In [186]:
```
get_cursor(user_query)
```

```
[{'count': 71049, '_id': 'Teddy73'}, {'count': 32611, '_id': 'jotp
e'}, {'count': 9617, '_id': 'catweazle67'}, {'count': 8669, '_id':
'cgn1234'}, {'count': 8590, '_id': 'okilimu'}]
```

In [194]:
```
# count number of shops
shop_query = db.aggregate([
        {"$match": {"shop": {"$exists": 1}}},
        {"$group": {"_id": None, "count": {"$sum": 1}}}
    ])
```

In [195]:
```
get_cursor(shop_query)
```

```
[{'count': 453, '_id': None}]
```

In [212]:
```
# count number of documents in cologne-ehrenfeld district (the dist
rict I used to live in)
ehrenfeld_query = db.aggregate([
        {"$match": {"address.postcode": "50823"}},
        {"$group": {"_id": None, "count": {"$sum": 1}}}
    ])
```

In [213]:
```
get_cursor(ehrenfeld_query)
```

```
[{'count': 308, '_id': None}]
```

Additional ideas

- What are the top 5 amenities?
- Which are the top 3 cuisines?
- Which historic sites exist in Cologne?
- What are the top 5 shops?
- Which leisure venues exist in Cologne?
- How many gay venues exist in Cologne?

In [274]:
```
# top 5 amenities in Cologne
amenity_query = db.aggregate([
        {"$match": {"amenity": {"$exists": 1}}},
        {"$group": {"_id": "$amenity", "count" : {"$sum": 1}}},
        {"$sort": {"count": -1}},
        {"$limit" : 5}
    ])
```

Cologne seems to be a city where parking is difficult, since the top amenity is considered parking (with bicycle parking ath the fourth position). Further, apart from restaurants and benches, post boxes are listed

In [275]:
```
get_cursor(amenity_query)
```

```
[{'count': 235, '_id': 'parking'}, {'count': 151, '_id': 'bench'},
{'count': 115, '_id': 'restaurant'}, {'count': 69, '_id': 'bicycle
_parking'}, {'count': 61, '_id': 'post_box'}]
```

In [277]:
```
# top 3 cuisines in Cologne
cuisine_query = db.aggregate([
        {"$match": {"cuisine": {"$exists": 1}}},
        {"$group": {"_id": "$cuisine", "count" : {"$sum": 1}}},
        {"$sort": {"count": -1}},
        {"$limit" : 3}
    ])
```

Although only a few documents contain cuisine tags, the result is not unusual for Cologne. Being a city with a numerous Italian and Turkish community, top three cuisines are considered Italian, Turkish (Kebab is my proxy for Turkish here) and German.

In [278]:
```
get_cursor(cuisine_query)
```

```
[{'count': 14, '_id': 'italian'}, {'count': 8, '_id': 'kebab'}, {'
count': 8, '_id': 'german'}]
```

In [228]:
```
# Which historic sites exist in Cologne?
historic_query = db.aggregate([
        {"$match": {"historic": {"$exists": 1}}},
        {"$group": {"_id": "$historic", "count" : {"$sum": 1}}},
        {"$sort": {"count": -1}}
    ])
```

The most frequent historic site is labeled memorial, probably related to world war 2

In [229]:
```
get_cursor(historic_query)
```

```
[{'count': 164, '_id': 'memorial'}, {'count': 66, '_id': 'heritage
'}, {'count': 5, '_id': 'wayside_cross'}, {'count': 3, '_id': 'for
t'}, {'count': 2, '_id': 'yes'}, {'count': 2, '_id': 'archaeologic
al_site'}, {'count': 2, '_id': 'wayside_shrine'}, {'count': 1, '_i
d': 'citywalls'}, {'count': 1, '_id': 'technical_monument'}, {'cou
nt': 1, '_id': 'building'}, {'count': 1, '_id': 'exhibit'}, {'coun
t': 1, '_id': 'castle'}, {'count': 1, '_id': 'city_gate'}, {'count
': 1, '_id': 'cemetery'}]
```

In [183]:
```
# Which are the top 5 shop categories in Cologne?
shop_query2 = db.aggregate([
        {"$match": {"shop": {"$exists": 1}}},
        {"$group": {"_id": "$shop", "count" : {"$sum": 1}}},
        {"$sort": {"count": -1}},
        {"$limit" : 5}
    ])
```

No suprises here, the kiosk a.k.a. "büdchen", which means small shop where you can buy alcohol late at night in German, is the top shop in Cologne

In [184]:
```
get_cursor(shop_query2)
```

```
[{'count': 65, '_id': 'kiosk'}, {'count': 45, '_id': 'clothes'}, {
'count': 29, '_id': 'hairdresser'}, {'count': 24, '_id': 'bakery'}
, {'count': 23, '_id': 'supermarket'}]
```

In [224]:
```
# Which leisure venues exist in Cologne?
leisure_query = db.aggregate([
        {"$match": {"leisure": {"$exists": 1}}},
        {"$group": {"_id": "$leisure", "count" : {"$sum": 1}}},
        {"$sort": {"count": -1}}
    ])
```

After consulting with Google translate, I learned that pitch is considered a synonym to playground. Thus having pitch as the most frequently listed leisure venue in Cologne makes sense

In [225]:
```
get_cursor(leisure_query)
```

```
[{'count': 64, '_id': 'pitch'}, {'count': 63, '_id': 'playground'}
, {'count': 39, '_id': 'park'}, {'count': 17, '_id': 'sports_centr
e'}, {'count': 12, '_id': 'garden'}, {'count': 12, '_id': 'outdoor
_seating'}, {'count': 6, '_id': 'swimming_pool'}, {'count': 4, '_i
d': 'horse_riding'}, {'count': 2, '_id': 'common'}, {'count': 2, '
_id': 'dog_park'}, {'count': 2, '_id': 'nature_reserve'}, {'count'
: 1, '_id': 'fitness_station'}, {'count': 1, '_id': 'dance'}, {'co
unt': 1, '_id': 'track'}, {'count': 1, '_id': 'tanning_salon'}, {'
count': 1, '_id': 'marina'}, {'count': 1, '_id': 'adult_gaming_cen
tre'}, {'count': 1, '_id': 'fitness_centre'}]
```

In [220]:
```
# How many gay venues exist in Cologne?
gay_query = db.aggregate([
        {"$match": {"gay": {"$exists": 1}}},
        {"$group": {"_id": None, "count" : {"$sum": 1}}},
    ])
```

Apparently there exist only 1 gay venue in our sample, which is cleary a strange result for Cologne (the capital of gays in Germany).

In [221]:
```
get_cursor(gay_query)
```

```
[{'count': 1, '_id': None}]
```

Conclusion

After my short review of Cologne's OSM data I am suprised of the general quality of the data, especially related to street names. Additionalyl OSM users tag documents which need to be fixed with corresponding tags (FIXME), which makes it easy for new users to contribute. Two problems exist which could be fixed easily (postal code and city tags), further opening times are recorded lacking a standard input format. Considering Cologne's status as the capital of gay people in Germany, it is suprising that only one document carries the tag "gay".