Open Street Map: A case study on the data

Author: Sagarnil Das

Date: 06/16/2017

Map Area:

City: New York

Link to Dataset: MapZen New York Dataset

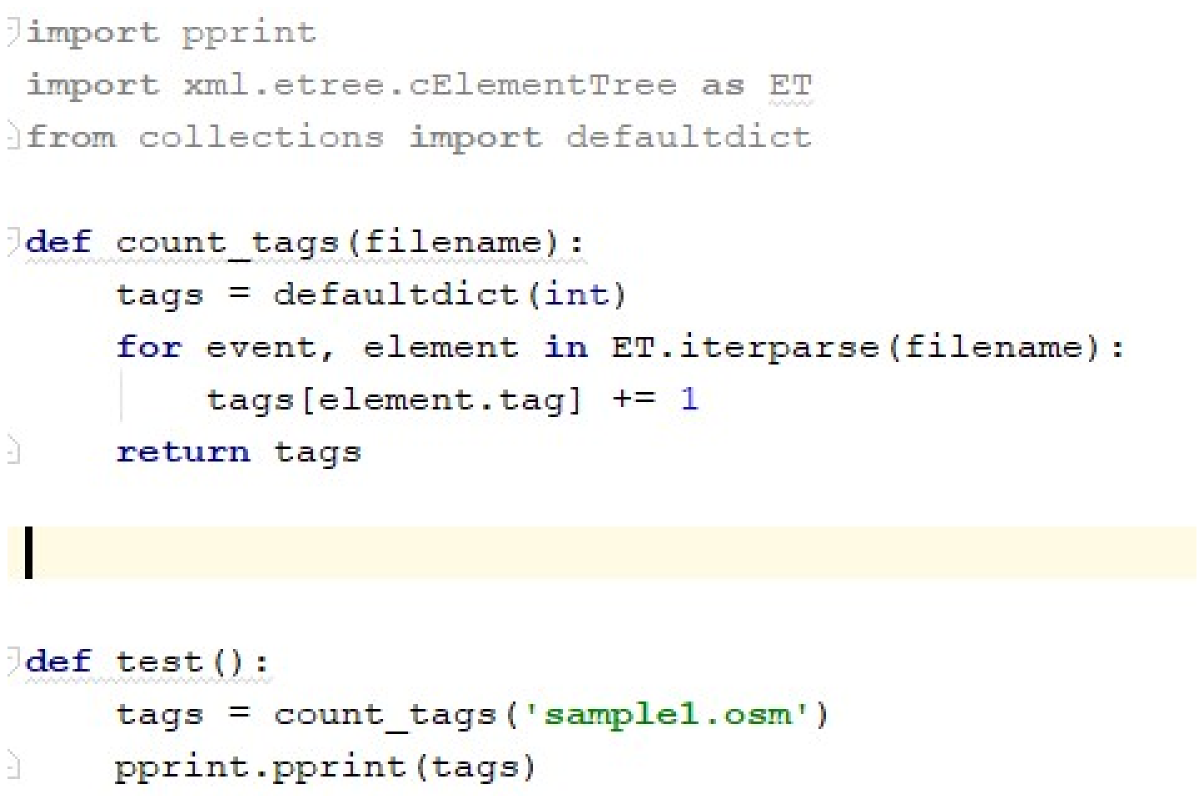
Though I am from India, I have lived a good years of my adult working life when I used to work for Department of Health. For some reason, I fell in love with this place. That's why I decided to work with the dataset of New York and wrangle and clean the data wherever necessary.

Dataset Information;

The original dataset was a huge file of 2.7 GB. So I used the sorten_osm.py to iterate through the file and write every 10th top level element. The resulting shorter file was 270 MB which I worked with. The code was provided in the instructors note.

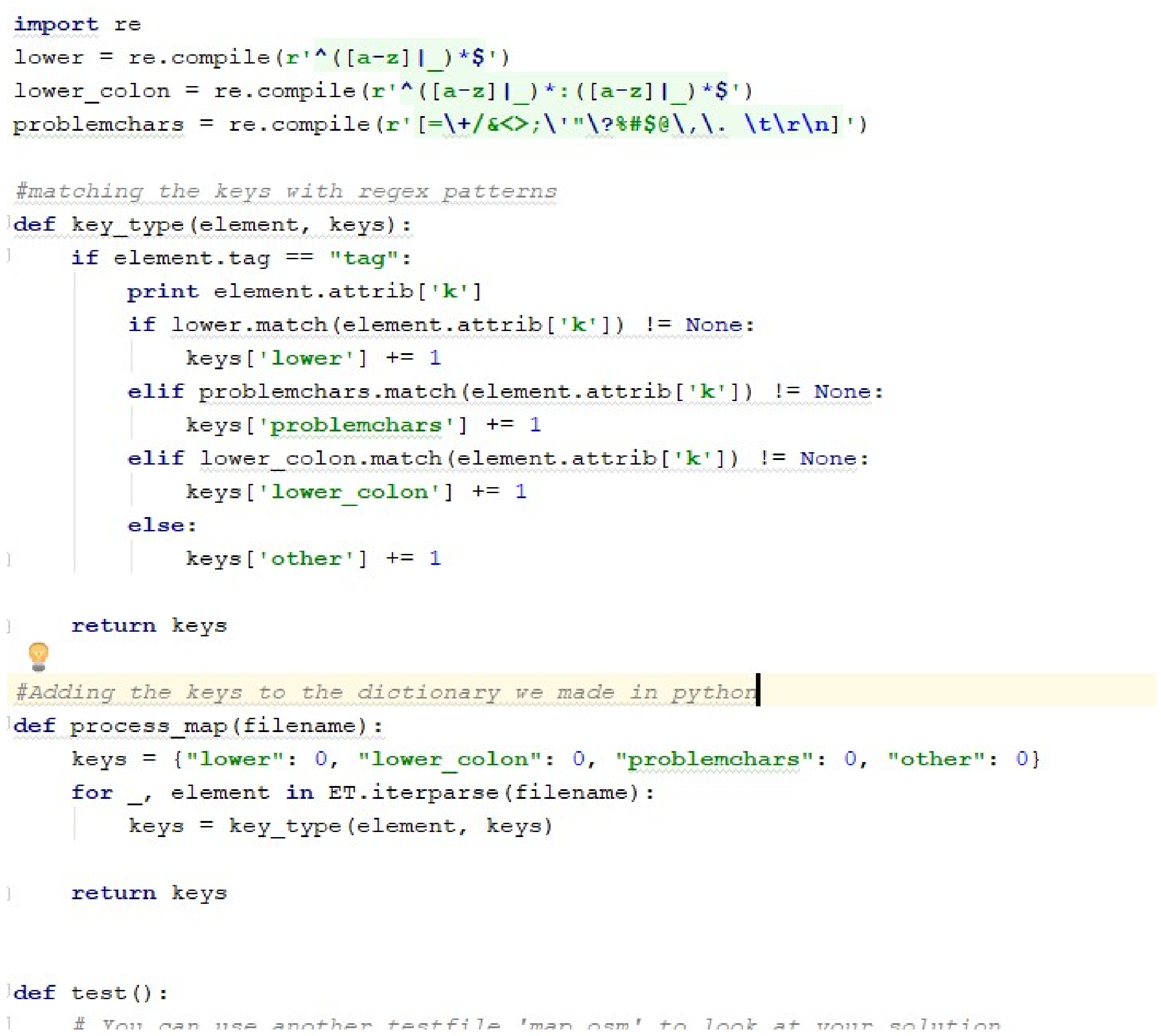
Steps in wrangling, cleaning and auditing the data:

1. **Initial Data Exploration:** Checked in the osm xml file how many unique tags are there to get a feeling of the data I am about to work with.



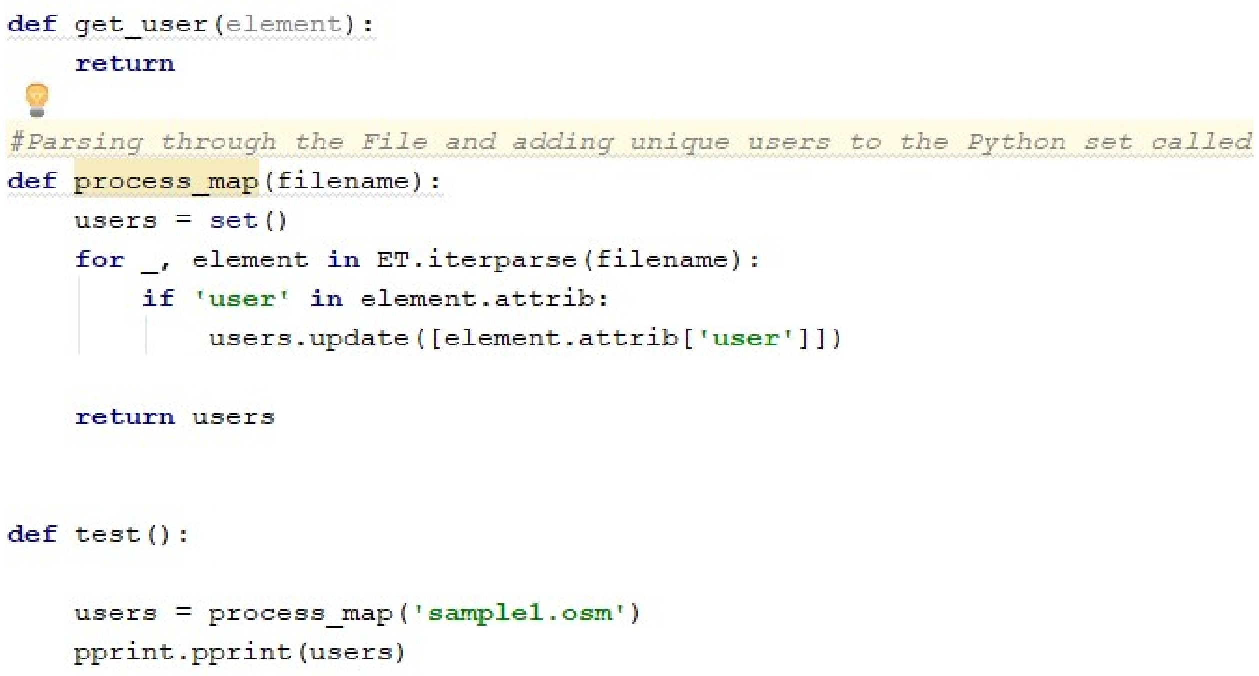
```
import pprint
import xml.etree.cElementTree as ET
from collections import defaultdict

def count_tags(filename):
    tags = defaultdict(int)
    for event, element in ET.iterparse(filename):
        tags[element.tag] += 1
    return tags


def test():
    tags = count_tags('sample1.osm')
    pprint.pprint(tags)
```

2. **Check for potential keys:** The 2nd step was to check for potential Keys inside the xml file as these keys would be required in future when we insert the clean data inside MongoDB. Inside the tags, there was an attribute called "k" and another called "v". They acted as a key value pair inside the xml file. So we used regex module to check for patterns inside the keys and look for any problematic characters.



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

#matching the keys with regex patterns
def key_type(element, keys):
    if element.tag == "tag":
        print element.attrib['k']
        if lower.match(element.attrib['k']) != None:
            keys['lower'] += 1
        elif problemchars.match(element.attrib['k']) != None:
            keys['problemchars'] += 1
        elif lower_colon.match(element.attrib['k']) != None:
            keys['lower_colon'] += 1
        else:
            keys['other'] += 1

    return keys

#Adding the keys to the dictionary we made in python
def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)

    return keys


def test():
    # You can use another testfile 'map.osm' to look at your solution
```

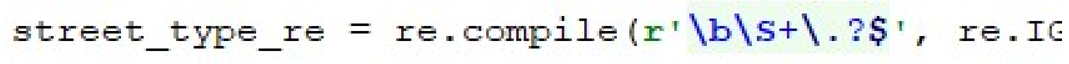
3. **Some more Data Exploration:** I did a little more data exploration again to get a good feel of the data. I coded to find out how many unique users contributed in developing the map in this particular area.



```
def get_user(element):
    return

#Parsing through the File and adding unique users to the Python set called
def process_map(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if 'user' in element.attrib:
            users.update([element.attrib['user']])

    return users


def test():

    users = process_map('sample1.osm')
    pprint.pprint(users)
```

Problems encountered in my Map:

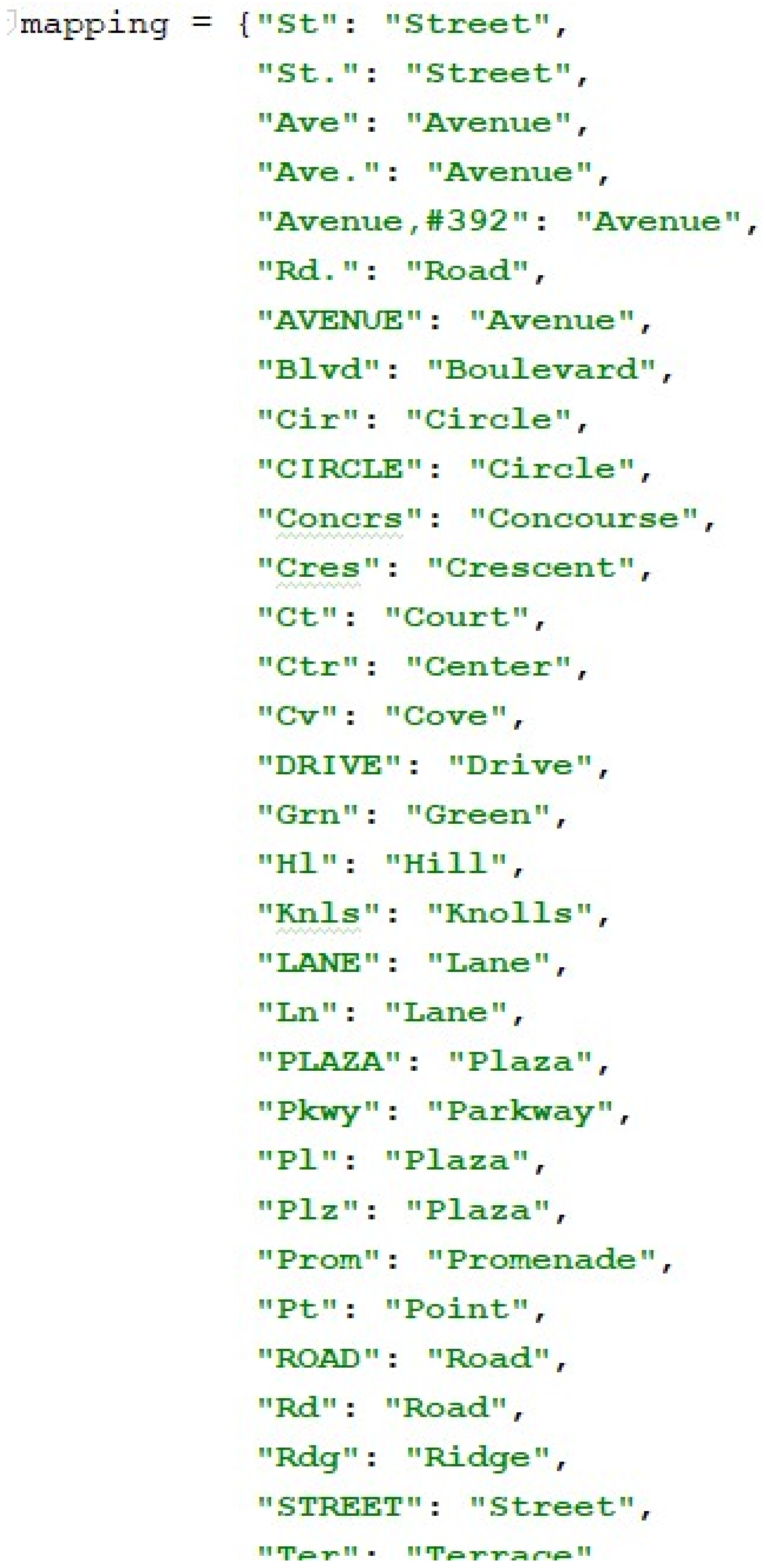
1. **Changing bad Street names:** It was noticed that in the many Street types had inconsistent names. A very high number of unique Street Types were observed(Street, Plaza, Court, Boulevard, Road etc). But the problem was the naming was inconsistent in many places. For example, in multiple places, instead of 'Street', it was written as 'St'. Same goes for 'Boulevard': 'Blvd', 'Parkway': 'Pkwy' and 'Avenue': 'Ave'. The types were too large. This is how this problem was handled. We wrote a regex to extract the street type from the address.

   ```
   street_type_re = re.compile(r'\b\S+\.?$', re.IG
   ```

   Then we made a list of some possible street types like 'Street', 'Boulevard', 'Drive', 'Road', 'Court' etc.

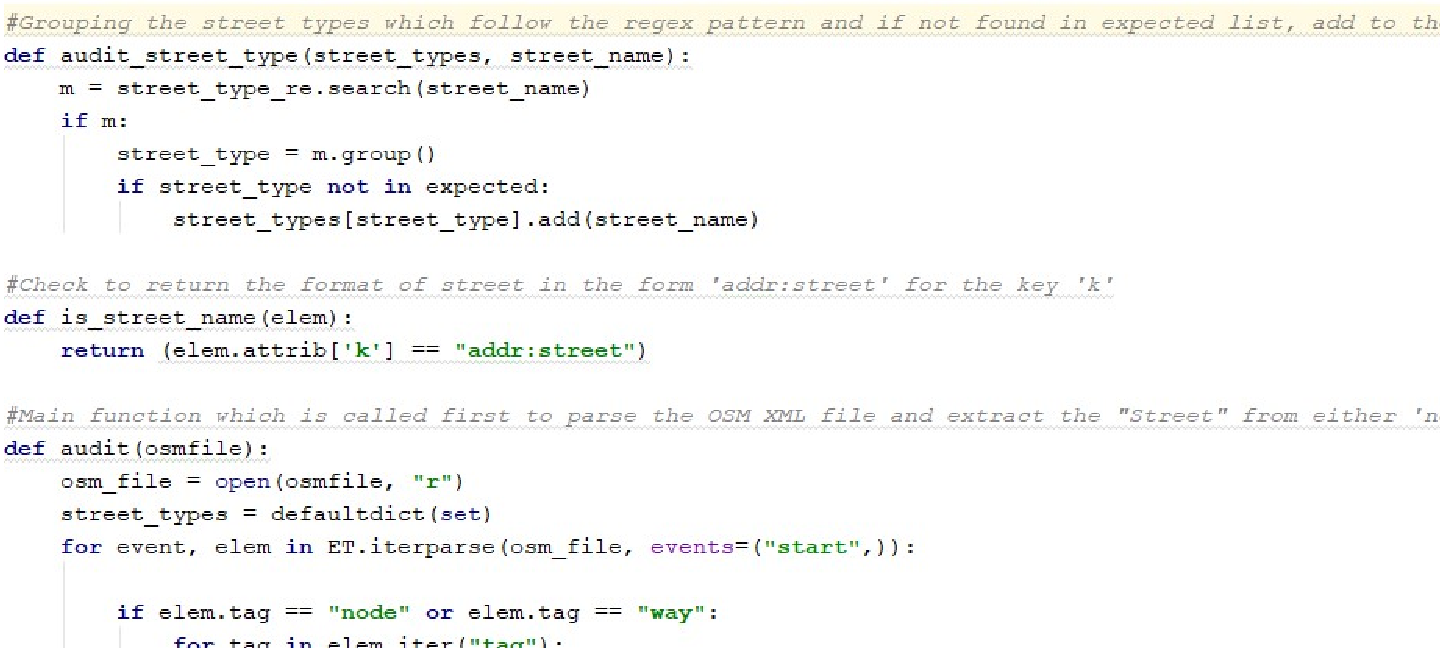
   We also made a mapper dictionary for short abbreviations used in many places, to map them to their full form.



```
mapping = {"St": "Street",
           "St.": "Street",
           "Ave": "Avenue",
           "Ave.": "Avenue",
           "Avenue,#392": "Avenue",
           "Rd.": "Road",
           "AVENUE": "Avenue",
           "Blvd": "Boulevard",
           "Cir": "Circle",
           "CIRCLE": "Circle",
           "Concrs": "Concourse",
           "Cres": "Crescent",
           "Ct": "Court",
           "Ctr": "Center",
           "Cv": "Cove",
           "DRIVE": "Drive",
           "Grn": "Green",
           "Hl": "Hill",
           "Knls": "Knolls",
           "LANE": "Lane",
           "Ln": "Lane",
           "PLAZA": "Plaza",
           "Pkwy": "Parkway",
           "Pl": "Plaza",
           "Plz": "Plaza",
           "Prom": "Promenade",
           "Pt": "Point",
           "ROAD": "Road",
           "Rd": "Road",
           "Rdg": "Ridge",
           "STREET": "Street",
           "Ter": "Terrace"
```

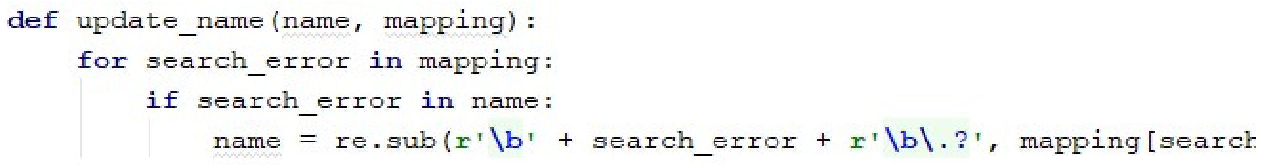
This mapping was not built in one go. With an initial mapping, we first built the functions and saw the possible street types and then we modified this mapping to add whatever values we didn't add the first time. Apart from that, these were the functions we wrote for cleaning the street data.

a) For the auditing purpose these three functions were written:



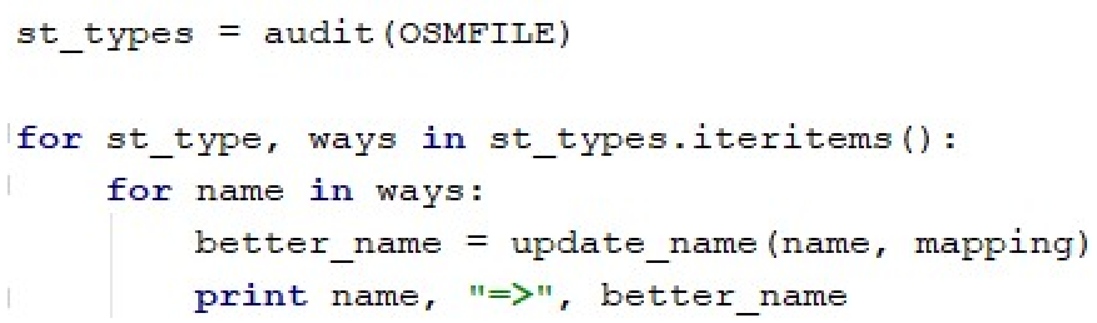
```
#Grouping the street types which follow the regex pattern and if not found in expected list, add to th
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)

#Check to return the format of street in the form 'addr:street' for the key 'k'
def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")

#Main function which is called first to parse the OSM XML file and extract the "Street" from either 'n
def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
```

b) For updating the name from the abbreviation to the full name, this function was written:



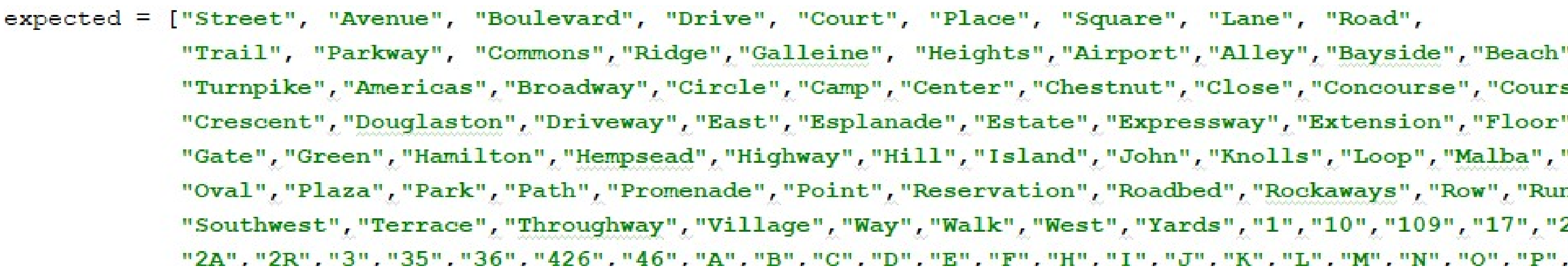
```
def update_name(name, mapping):
    for search_error in mapping:
        if search_error in name:
            name = re.sub(r'\b' + search_error + r'\b\.?', mapping[searck
```

But now we faced some problems. First I ran the audit function on the OSM file, it worked. After that I tried updating the names like this:



```
st_types = audit(OSMFILE)

for st_type, ways in st_types.iteritems():
    for name in ways:
        better_name = update_name(name, mapping)
        print name, "=>", better_name
```

For the 2nd part of the code, that is updating the names, I got a strange error ‘KeyError’: ’Certain_value’ not found. Upon further investigation, I discovered what the problem was. My list of possible city names was not complete. I just gave some initial names. But if while getting the street, it doesn’t encounter the same value in the ‘expected’ list, python thinks that it does not exist. So I went through all the street types in the dictionary I created in python and added all the street types inside the list. Note: Many street names were not incorrect but they came up in my audit as they followed the same pattern. For example: ‘East Parkway 4’. So to account for them and the rest, this was my final list:



```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
             "Trail", "Parkway", "Commons","Ridge","Galleine", "Heights","Airport","Alley","Bayside","Beach"
             "Turnpike","Americas","Broadway","Circle","Camp","Center","Chestnut","Close","Concourse","Cours
             "Crescent","Douglaston","Driveway","East","Esplanade","Estate","Expressway","Extension","Floor"
             "Gate","Green","Hamilton","Hempsead","Highway","Hill","Island","John","Knolls","Loop","Malba","
             "Oval","Plaza","Park","Path","Promenade","Point","Reservation","Roadbed","Rockaways","Row","Rur
             "Southwest","Terrace","Throughway","Village","Way","Walk","West","Yards","1","10","109","17","2
             "2A"."2R"."3"."35"."36"."426"."46"."A"."B"."C"."D"."E"."F"."H"."I"."J"."K"."L"."M"."N"."O"."P".
```

Next Steps:

1. **Data wrangling and making the data shape correctly so that we can insert the data into MongoDB:** The next task was to wrangle and transform the data into this type of format:

```
{
"id": "2406124091",
"type: "node",
"visible":"true",
"created": {
          "version":"2",
          "changeset":"17206049",
          "timestamp":"2013-08-03T16:43:42Z",
          "user":"linuxUser16",
          "uid":"1219059"
        },
"pos": [41.9757030, -87.6921867],
"address": {
          "housenumber": "5157",
          "postcode": "60625",
          "street": "North Lincoln Ave"
        },
"amenity": "restaurant",
"cuisine": "mexican",
"name": "La Cabana De Don Luis",
"phone": "1 (773)-271-5176"
}
```

The code is given in project_3_file_1.py. After wrangling and transforming the data, we wrote the data into a JSON file which we will now import into MongoDB.

2. **Data import to MongoDB:**

```
mongoimport --db osm --collection newyorkosm --type json --file
sample3.json
```

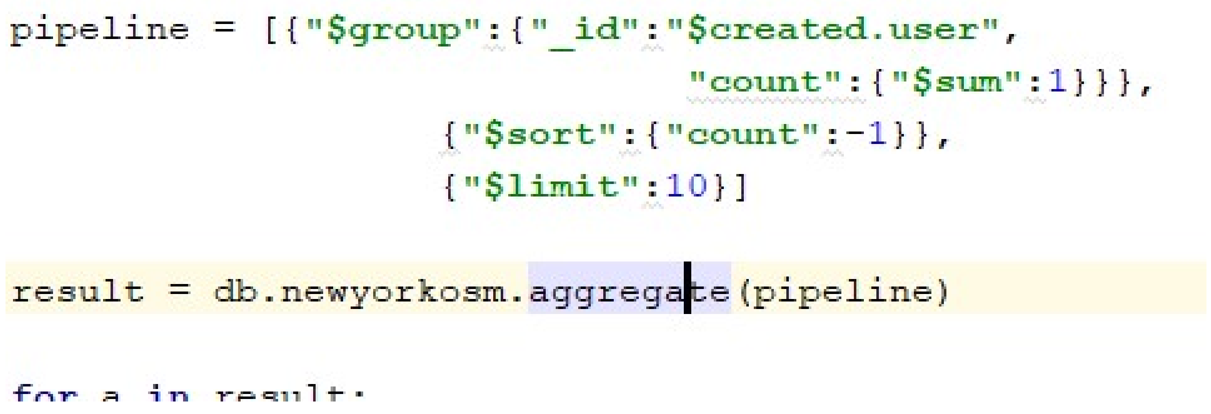
Data Overview:

1. **File Size:**
   New-york_new-york.osm – original OSM file (Size = 2.7 GB)
   Sample1.osm – Shortened OSM file (Size = 278 MB)
   Sample3.json – Final cleaned JSON file (Size = 514 MB)

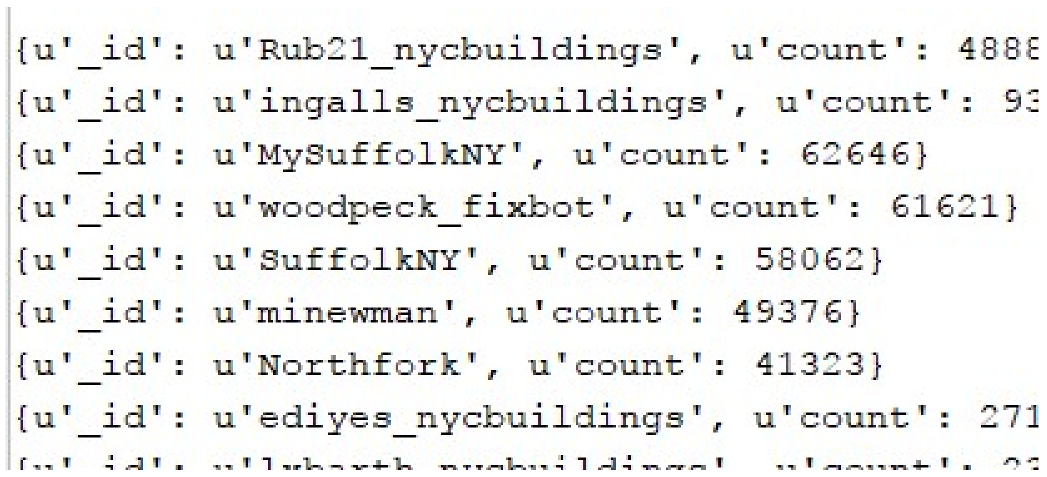
2. **MongoDB overview:**
   a) Number of documents: 1330111 [db.newyorkosm.find().count()]
   b) Number of Nodes: 1149904 [db.newyorkosm.find({“type”:”node”}).count()]
   c) Number of Ways: 180177 [db.newyorkosm.find({“type”:”way”}).count()]
   d) Number of Uniques users: 2615 [print len(db.newyorkosm.distinct("created.user"))]

Other ideas about the dataset:

1. Top 10 contributing users for this particular map:

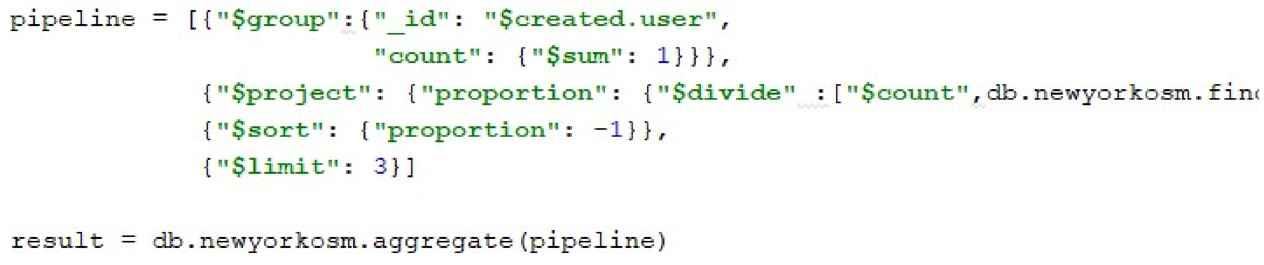


```
pipeline = [{"$group":{"_id":"$created.user",
                                          "count":{"$sum":1}}},
                        {"$sort":{"count":-1}},
                        {"$limit":10}]

result = db.newyorkosm.aggregate(pipeline)

for a in result:
```

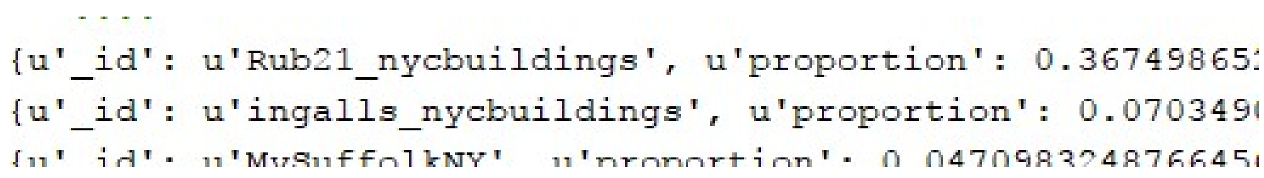


```
{u'_id': u'Rub21_nycbuildings', u'count': 488
{u'_id': u'ingalls_nycbuildings', u'count': 93
{u'_id': u'MySuffolkNY', u'count': 62646}
{u'_id': u'woodpeck_fixbot', u'count': 61621}
{u'_id': u'SuffolkNY', u'count': 58062}
{u'_id': u'minewman', u'count': 49376}
{u'_id': u'Northfork', u'count': 41323}
{u'_id': u'ediyes_nycbuildings', u'count': 271
{u'_id': u'lwbarth_nycbuildings', u'count': 23
```

2. Proportion of top users contribution:

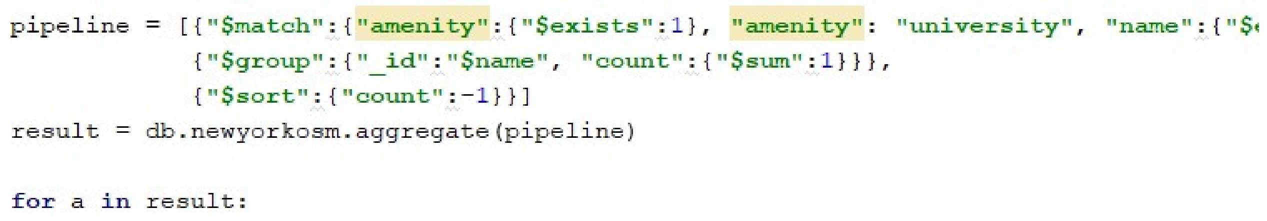


```
pipeline = [{"$group":{"_id": "$created.user",
                          "count": {"$sum": 1}}},
             {"$project": {"proportion": {"$divide" :["$count",db.newyorkosm.fin
             {"$sort": {"proportion": -1}},
             {"$limit": 3}]

result = db.newyorkosm.aggregate(pipeline)
```

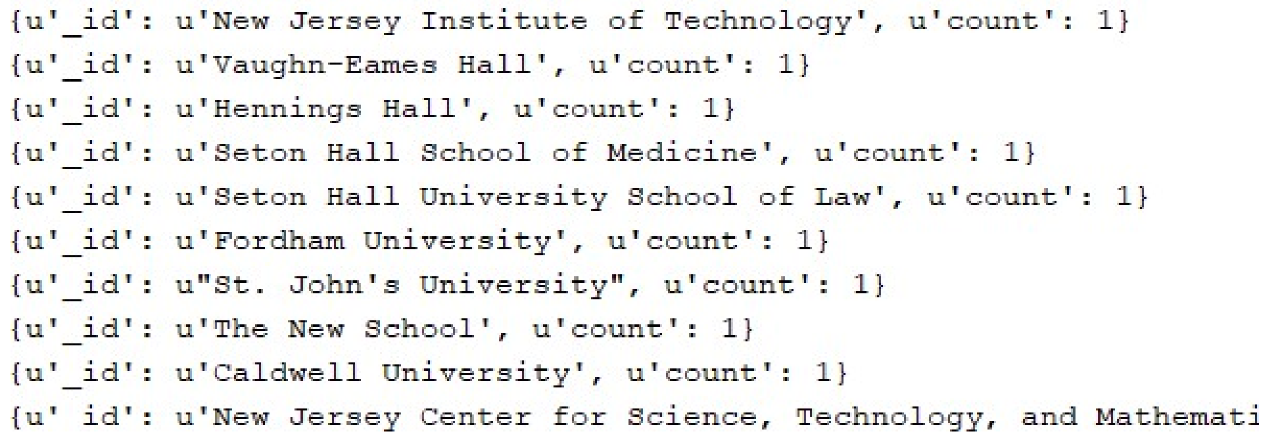


```
    ....
{u'_id': u'Rub21_nycbuildings', u'proportion': 0.36749865
{u'_id': u'ingalls_nycbuildings', u'proportion': 0.070349
{u'_id': u'MySuffolkNY', u'proportion': 0.047098324876645
```

3. List of Universities:

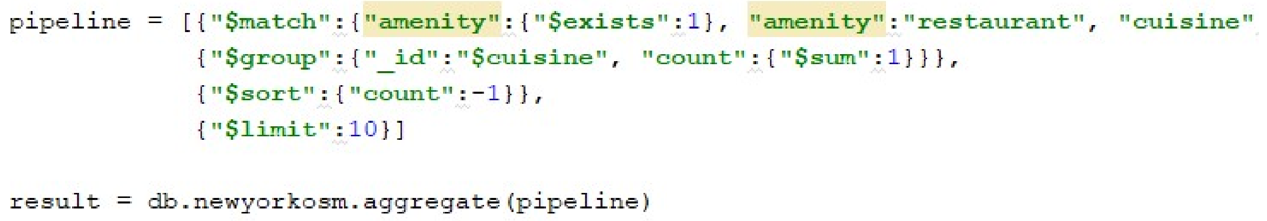


```
pipeline = [{"$match":{"amenity":{"$exists":1}, "amenity": "university", "name":{"$
            {"$group":{"_id":"$name", "count":{"$sum":1}}},
            {"$sort":{"count":-1}}]
result = db.newyorkosm.aggregate(pipeline)

for a in result:
```

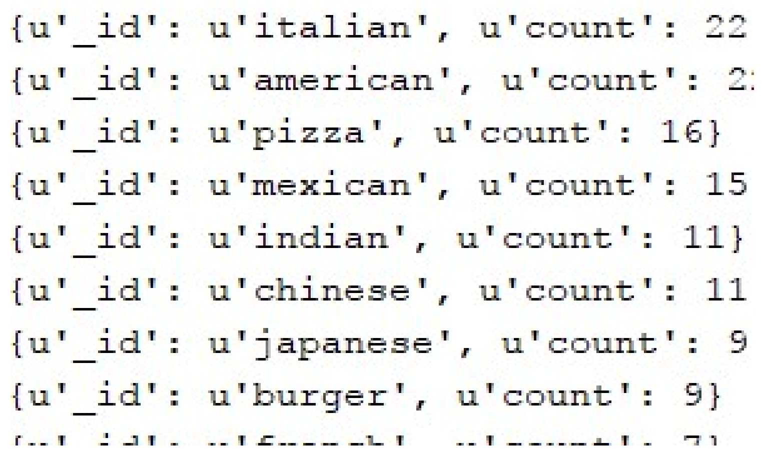


```
{u'_id': u'New Jersey Institute of Technology', u'count': 1}
{u'_id': u'Vaughn-Eames Hall', u'count': 1}
{u'_id': u'Hennings Hall', u'count': 1}
{u'_id': u'Seton Hall School of Medicine', u'count': 1}
{u'_id': u'Seton Hall University School of Law', u'count': 1}
{u'_id': u'Fordham University', u'count': 1}
{u'_id': u"St. John's University", u'count': 1}
{u'_id': u'The New School', u'count': 1}
{u'_id': u'Caldwell University', u'count': 1}
{u' id': u'New Jersey Center for Science, Technology, and Mathemati
```

4. Most Popular Cuisines:

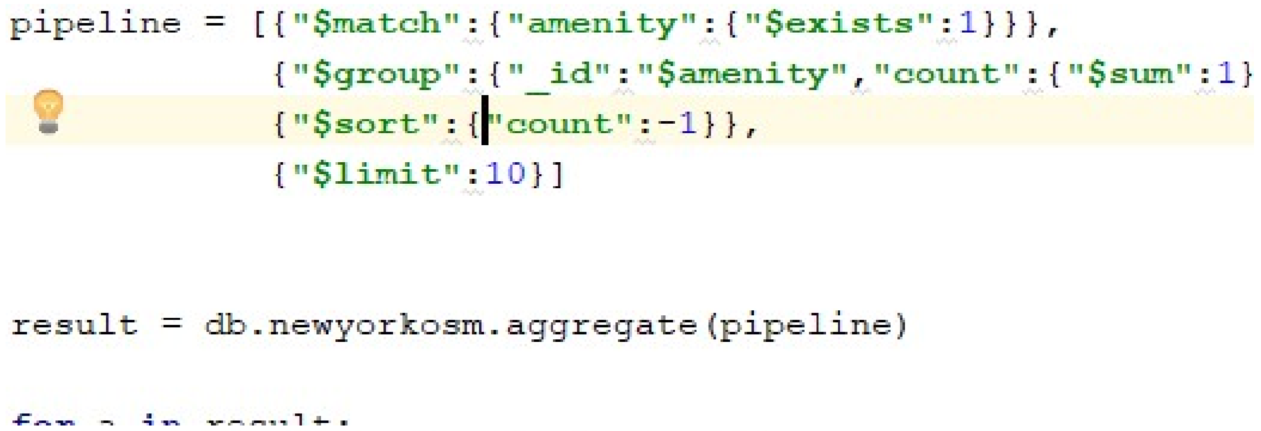


```
pipeline = [{"$match":{"amenity":{"$exists":1}, "amenity":"restaurant", "cuisine"
            {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
            {"$sort":{"count":-1}},
            {"$limit":10}]

result = db.newyorkosm.aggregate(pipeline)
```

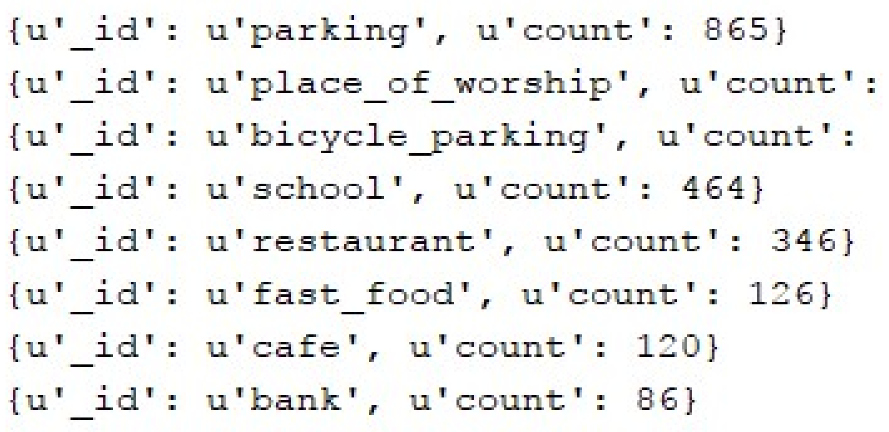


```
{u'_id': u'italian', u'count': 22
{u'_id': u'american', u'count': 2
{u'_id': u'pizza', u'count': 16}
{u'_id': u'mexican', u'count': 15
{u'_id': u'indian', u'count': 11}
{u'_id': u'chinese', u'count': 11
{u'_id': u'japanese', u'count': 9
{u'_id': u'burger', u'count': 9}
{u'_id': u'french', u'count': 7}
```

5. Top 10 amenities:



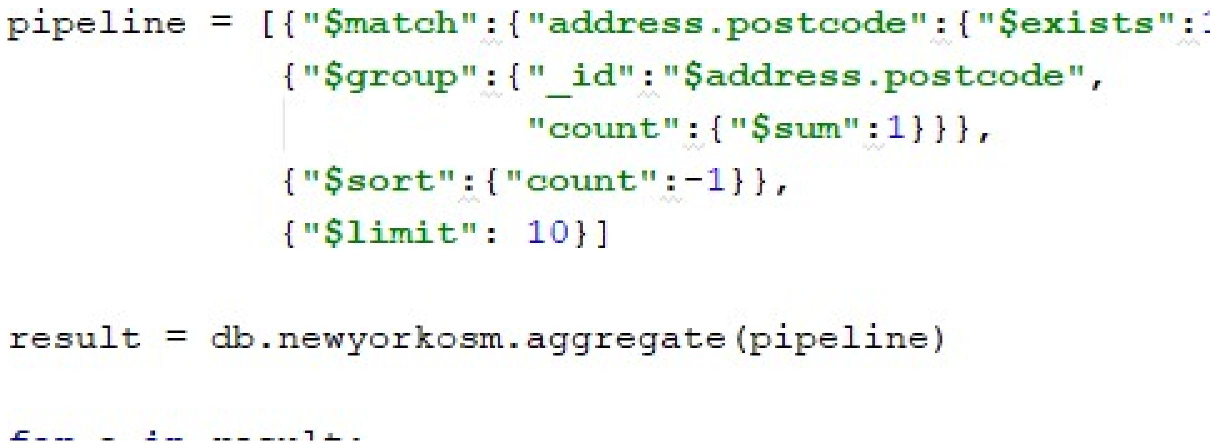
```
pipeline = [{"$match":{"amenity":{"$exists":1}}},
            {"$group":{"_id":"$amenity","count":{"$sum":1}
            {"$sort":{"count":-1}},
            {"$limit":10}]


result = db.newyorkosm.aggregate(pipeline)

for a in result:
```



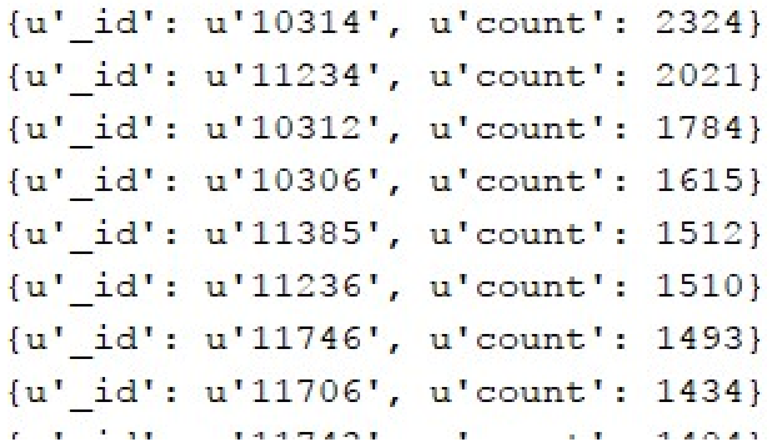
```
{u'_id': u'parking', u'count': 865}
{u'_id': u'place_of_worship', u'count':
{u'_id': u'bicycle_parking', u'count':
{u'_id': u'school', u'count': 464}
{u'_id': u'restaurant', u'count': 346}
{u'_id': u'fast_food', u'count': 126}
{u'_id': u'cafe', u'count': 120}
{u'_id': u'bank', u'count': 86}
```

No surprises here!... ☺

6. Top 10 postal codes:

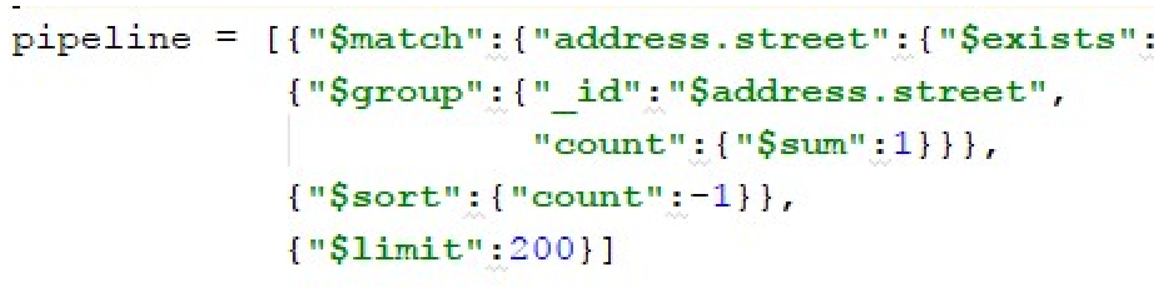


```
pipeline = [{"$match":{"address.postcode":{"$exists":
            {"$group":{"_id":"$address.postcode",
                       "count":{"$sum":1}}},
            {"$sort":{"count":-1}},
            {"$limit": 10}]


result = db.newyorkosm.aggregate(pipeline)

for a in result:
```

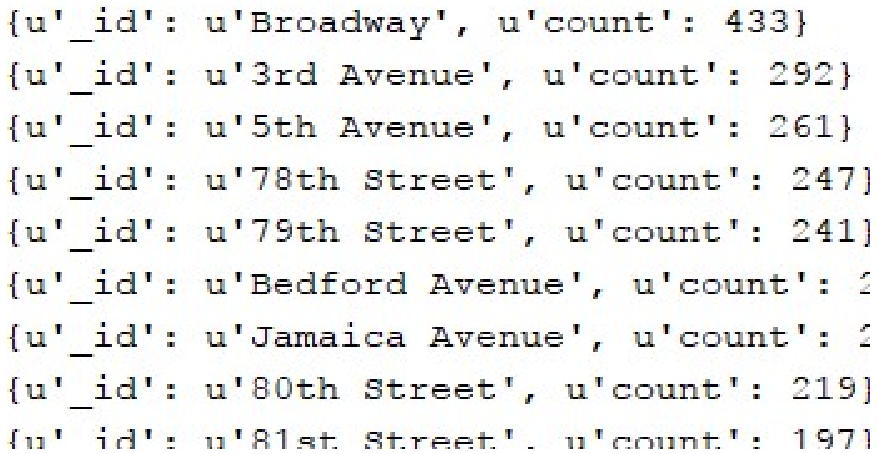


```
{u'_id': u'10314', u'count': 2324}
{u'_id': u'11234', u'count': 2021}
{u'_id': u'10312', u'count': 1784}
{u'_id': u'10306', u'count': 1615}
{u'_id': u'11385', u'count': 1512}
{u'_id': u'11236', u'count': 1510}
{u'_id': u'11746', u'count': 1493}
{u'_id': u'11706', u'count': 1434}
```

7. Top 10 popular streets:



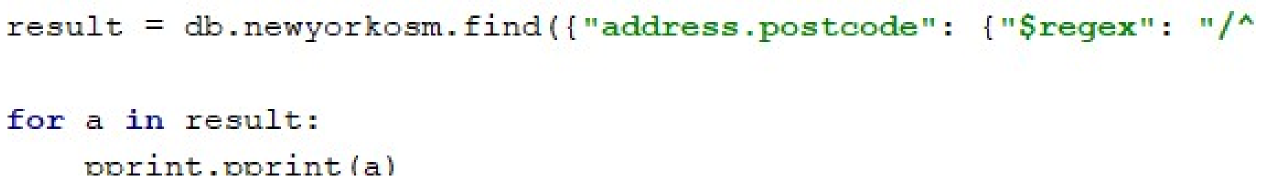
```
pipeline = [{"$match":{"address.street":{"$exists":
            {"$group":{"_id":"$address.street",
                       "count":{"$sum":1}}},
            {"$sort":{"count":-1}},
            {"$limit":200}]

result = db.newyorkosm.aggregate(pipeline)

for a in result:
```



```
{u'_id': u'Broadway', u'count': 433}
{u'_id': u'3rd Avenue', u'count': 292}
{u'_id': u'5th Avenue', u'count': 261}
{u'_id': u'78th Street', u'count': 247}
{u'_id': u'79th Street', u'count': 241}
{u'_id': u'Bedford Avenue', u'count': 2
{u'_id': u'Jamaica Avenue', u'count': 2
{u'_id': u'80th Street', u'count': 219}
{u'_id': u'81st Street', u'count': 197}
```

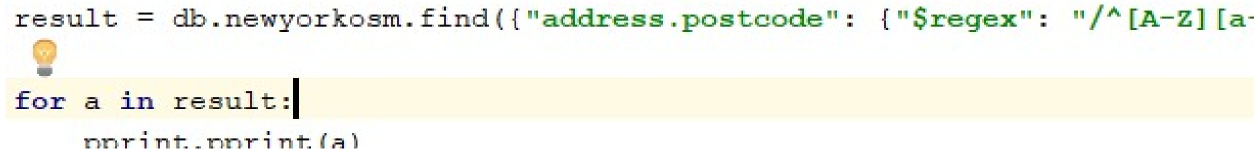
Checking for other potential problems:

1. Checking for Alphanumeric postal codes: (Result = 0)



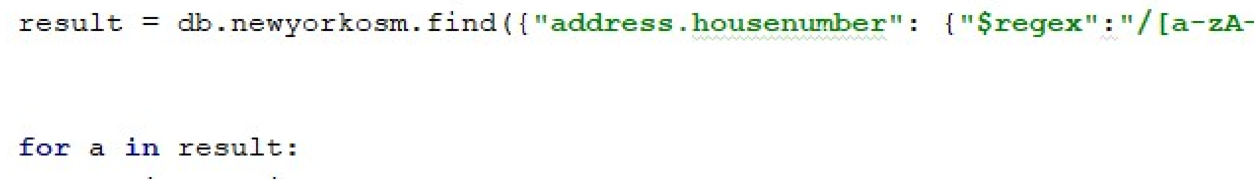
```
result = db.newyorkosm.find({"address.postcode": {"$regex": "/^

for a in result:
    pprint.pprint(a)
```

2. Checking for postal codes that start with an uppercase letter followed by onne or more lowercase letter: (Result = 0)



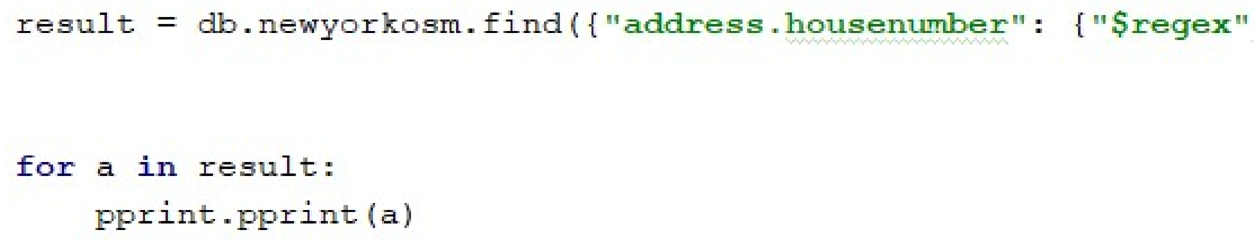
```
result = db.newyorkosm.find({"address.postcode": {"$regex": "/^[A-Z][a-

for a in result:
    pprint.pprint(a)
```

3. Checking for house numbers that has alphabetical characters: (Result = 0)



```
result = db.newyorkosm.find({"address.housenumber": {"$regex":"/[a-zA-


for a in result:
```

4. Checking for House characters that has spaces: (Result = 0)



```
result = db.newyorkosm.find({"address.housenumber": {"$regex"


for a in result:
    pprint.pprint(a)
```

So fortunately, all these additional pattern checks did not bring me any more erroneous values. So right now, the data which resides in my MongoDB is quite clean and has passed my auditing.

Conclusion:

So, during this whole process of gathering, extracting, cleaning and storing of our data, the biggest problem was fixing the street names with an appropriate street type. The dataset was pretty big and the huge number of street types astounded me. I never would have thought about it. But by doing a cyclical auditing, I was finally able to fix all the names. I did some additional failure checks also all of whose results came back as negative. So even though in my opinion, there are still chances of improvement, I believe the data is sufficiently cleaned for the purpose of this project of Data Wrangling. The additional queries performed on the database documents gave consistent results which supports the theory of clean data. I loved doing this project for the city which was practically my home for 3 years.

References:

1. Maps: https://www.openstreetmap.org/
2. Maps: https://mapzen.com/data/metro-extracts/
3. Regexes: https://github.com/SamMorrowDrums/Udacity-OpenStreetmap
4. Project Layout: Udacity sample project