Data Wrangling with Open Street Map using Python and MongoDB

Map Area: Berkeley, California, USA

The data was downloaded using the Open Street Map API:
http://overpass-api.de/query_form.html
And entering the coordinates into the “Overpass API Query Form”:
(node(37.7983,-122.3504,37.8873,-122.1929);<;);out meta;

Raw data: berkeley.osm: 50.3mb
Json file with cleaned data: berkeley.osm.json: 67.8mb

Data Auditing and Cleaning

Using the python scripts in “data wrangling map data audit.ipynb” and “data wrangling cleaning.ipynb”, I audited, then cleaned and wrote the cleaned data to a json file.

- I created sub-dictionaries for the fields “created” and “address”, as well as for fields denoted by “:”, such as “caltrans:district”. If there were more nested fields beyond this, such as “level1:level2:level3”, I would flatten out further levels by replacing the “:” with a “_”. For example, the dictionary would be { “level1” : {“level2_level3”: “value”}. Also, some nested fields had the same name as non-nested fields. For example a single xml element had both “toilets” and “toilets:wheelchair”. To avoid overriding one or the other, a field that was also a dictionary, I append the suffix “_collection”. So “toilets:wheelchair” becomes { “toilets_collection”: { “wheelchair”: “value”}}. Similarly, some fields were decided by design, such as “type”, which refers to whether it’s a “way” or “node”. When that same field was present in the data itself, I appended “_”, so “type_” would not override the original “type” field.
- Street usually contains names, but sometimes start with the house number, such as “111 Grand Ave”. I save just the street name, and if there is not already a house number field, I save the house number. I also check for street names that are numbered, such as 1st, 2nd, 3rd, 4th, 5th, 12345th street etc, and do not save those as house numbers.
- Post code usually contains a five-digit number, such as “94109”, but sometimes has the state and/or post code extension, such as “CA 94607” or “94612-2202”. I parse out the state and post code extension, and if the field does not already exist, save the state and extension fields. The main five-digit number is saved to the post code field.
- I also search for and replace abbreviations, such as “Pl” replaced with “Plaza”, and “Sq.” replaced with “Square”, or “Btwn” replaced with “Between”.
- The amenity contains location descriptions, some of which I standardized. For instance, “parking_space” and “parking_entrance” are standardized to the more common “parking”. Similarly, operator contains names of businesses or institutions, which I likewise

standardized. For instance, “UC Berkeley”, “University California Berkeley” are all standardized to “University of California, Berkeley (Cal)”.

- Lanes were all convertible into integers, so I saved those as integers to allow for aggregations, such as finding the average number of lanes for a highway.

Data Overview

File Sizes:
berkeley.osm: 50.3mb
berkeley.osm.json: 67.8mb

Number of documents:
db.berkeley.count()
231784

- Number of element types (node or way):

```
query = [{"$group":
              {"_id": "$type",
               "count":{"$sum":1}}},
       {"$sort": {"count": -1}}
      ]
result = [c for c in db.berkeley.aggregate(query)]
pprint.pprint(result)
================================
[{u'_id': u'node', u'count': 201050},
 {u'_id': u'way', u'count': 30734}]
================================
```

- Number of unique users:

```
query = [{"$group": {
          "_id": "unique_users_id",
          "unique_users_set": {"$addToSet":"$created.user"}
       }
        },
        {"$limit": 1}
       ]
result =[c['unique_users_set'] for c in db.berkeley.aggregate(query)]
len(result[0])
================================
548
================================
```

- Top 5 users by numer of elements contributed:

```
query=[{"$group":{
          "_id":{"uid":"$created.uid","user":"$created.user"},
```

```
            "count":{"$sum": 1}
        }},
        {"$sort": {"count":-1}},
        {"$limit":5}
      ]
result = [c for c in db.berkeley.aggregate(query)]
Result
============================================================
[{u'_id': {u'uid': u'1249504', u'user': u'EranChazan'}, u'count': 35663},
 {u'_id': {u'uid': u'933797', u'user': u'oba510'}, u'count': 30197},
 {u'_id': {u'uid': u'381909', u'user': u'JessAk71'}, u'count': 21669},
 {u'_id': {u'uid': u'941449', u'user': u'lyzidiamond'}, u'count': 21489},
 {u'_id': {u'uid': u'153669', u'user': u'dchiles'}, u'count': 20120}]
============================================================
```

- Top five operators by count (most tags do not have this field)

```
query = [{"$group": {"_id": "$operator",
                     "count": {"$sum":1}}},
    {"$sort": {"count":-1}},
    {"$limit": 6}
]

cursor = db.berkeley.aggregate(query)
result = [c for c in cursor]
result
============================================================
[{u'_id': None, u'count': 231251},
 {u'_id': u'AC Transit', u'count': 239},
 {u'_id': u'City CarShare', u'count': 51},
 {u'_id': u'USPS', u'count': 31},
 {u'_id': u'University of California, Berkeley (Cal)', u'count': 20}]
============================================================
```

- Top five amenities:

```
query = [{"$group": {"_id": "$amenity",
                     "count": {"$sum":1}}},
    {"$sort": {"count":-1}},
    {"$limit": 6}
]
cursor = db.berkeley.aggregate(query)
result = [c for c in cursor]
pprint.pprint(result)
============================================================
[{u'_id': None, u'count': 227980},
 {u'_id': u'parking', u'count': 984},
 {u'_id': u'restaurant', u'count': 594},
 {u'_id': u'place_of_worship', u'count': 241},
```

```
 {u'_id': u'cafe', u'count': 237},
 {u'_id': u'bicycle_parking', u'count': 172}]
==========================================================
```

- Average number of lanes per way

```
query = [
   {"$match": {"type": "way"}},
   {"$group": {"_id": "all highways",
              "avg_lanes": {"$avg":"$lanes"}}
   },
   {"$limit": 5}

]
cursor = db.berkeley.aggregate(query)
result = [c for c in cursor]
pprint.pprint(result)
==========================================================
[{u'_id': u'all highways', u'avg_lanes': 2.6786060019361084}]
==========================================================
```

- Top five ways by number of nodes that each way refers to.

```
#most node references per way
query=[{"$match": {"type":"way"}},
       {"$unwind": "$node_refs"},
       {"$group": {"_id": {"id":"$id"},
                   "node_count": {"$sum": 1}}},
       {"$sort": {"node_count": -1}},
       {"$limit": 5}
      ]
cursor = db.berkeley.aggregate(query)
result = [c for c in cursor]
result
==========================================================
[{u'_id': {u'id': u'192565555'}, u'node_count': 711},
 {u'_id': {u'id': u'123075097'}, u'node_count': 438},
 {u'_id': {u'id': u'114224294'}, u'node_count': 370},
 {u'_id': {u'id': u'11564185'}, u'node_count': 352},
 {u'_id': {u'id': u'33088737'}, u'node_count': 290}]
==========================================================
```

By searching the map for the way ID, these top five ways are:
711 nodes
http://www.openstreetmap.org/way/192565555
The boundary around Redwood Regional Park and Anthony Chabot Regional Park.

438 nodes
http://www.openstreetmap.org/way/123075097
The boundary around Tilden Park

370 nodes
http://www.openstreetmap.org/way/114224294
The boundary around North Oakland Regional Sports Center

352 nodes
http://www.openstreetmap.org/way/11564185
The boundary around Lake Merritt in Oakland.

290 nodes
http://www.openstreetmap.org/way/33088737
The boundary around Piedmont neighborhood, Oakland.

- For each way, what is the average number of nodes that it references?

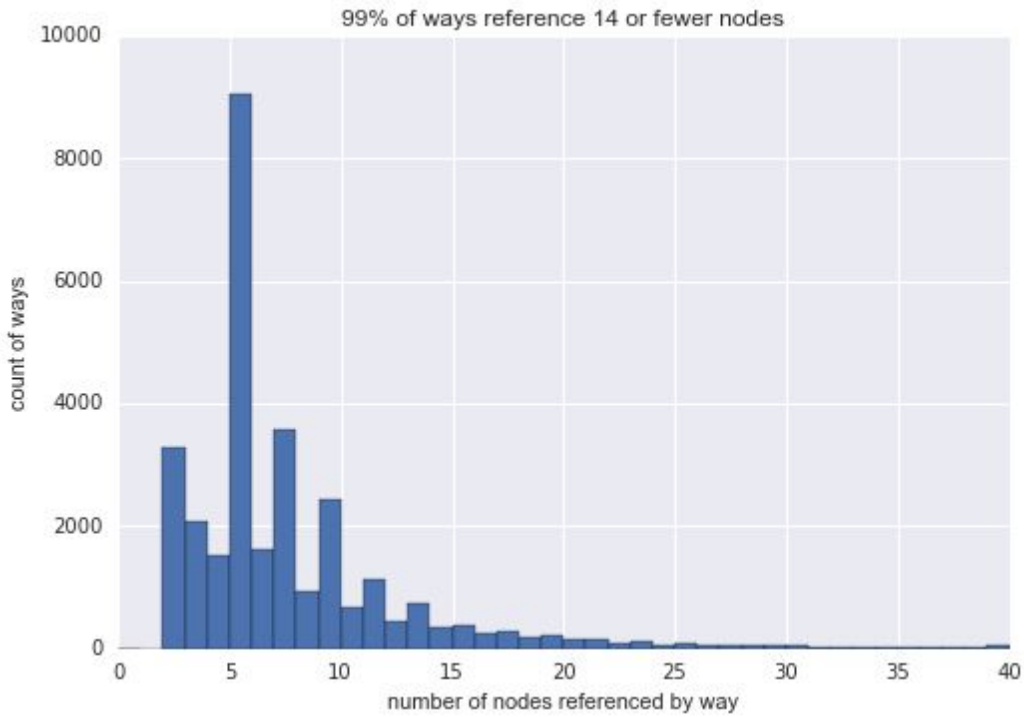
```
query=[{"$match": {"type":"way"}},
      {"$unwind": "$node_refs"},
      {"$group": {"_id": "$id",
                  "node_count": {"$sum": 1}}},
      {"$group": {"_id": "average nodes per way",
                  "avg": {"$avg": "$node_count"}
       }
      },
      #{"$sort": {"node_count": 1}},
      {"$limit": 5}
     ]
cursor = db.berkeley.aggregate(query)
result = [c for c in cursor]
result
```

```
==========================================================
[{u'_id': u'average nodes per way', u'avg': 8.082612090844016}]
==========================================================
```

- Most ways refer to fewer than 14 nodes

```
query=[{"$match": {"type":"way"}},
      {"$unwind": "$node_refs"},
      {"$group": {"_id": {"id":"$id"},
                  "node_count": {"$sum": 1}}},
      {"$sort": {"node_count": 1}},
      {"$project": {"_id":0,"node_count":1}}
     ]
cursor = db.berkeley.aggregate(query)
result = [c['node_count'] for c in cursor]
node_count_arr = np.array(result)
```

```
(q25,q50,q75,q90,q99) = np.percentile(node_count_arr,[25,50,75,90,99])
=======================================================
print "25% {} 50% {} 75% {} 99% {}".format(q25,q50,q75,q90,q99)
=======================================================

plt.hist(node_count_arr,bins=40,range=(0,40));
plt.title('99% of ways reference 14 or fewer nodes');
plt.xlabel("number of nodes referenced by way");
plt.ylabel("count of ways");
```

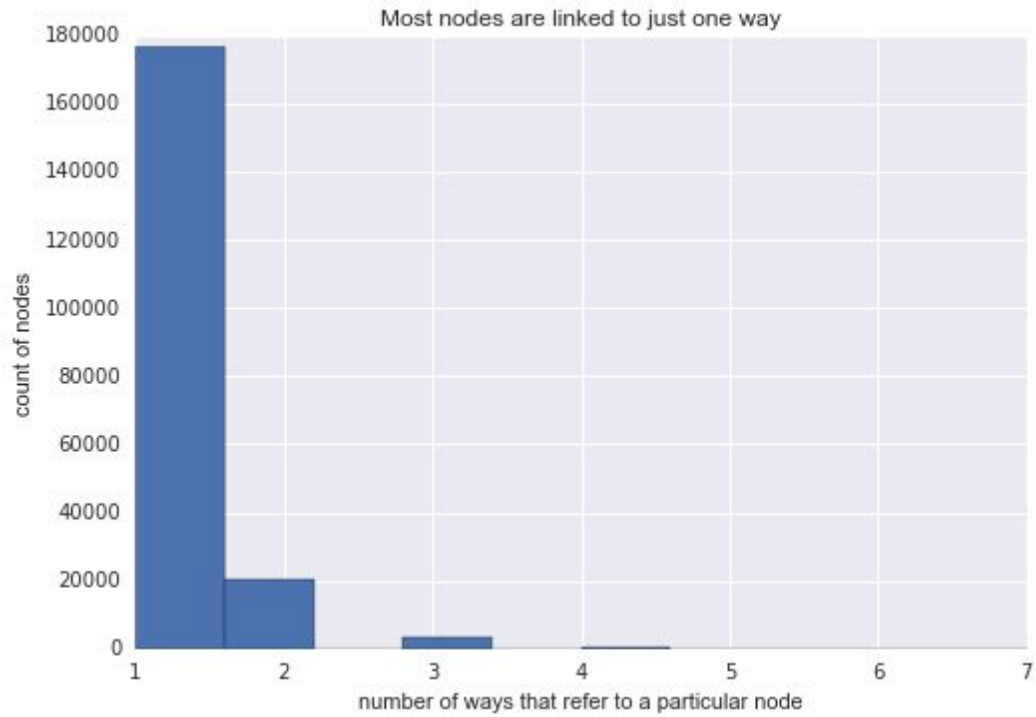


- How many ways are associated with each node?

```
"""
We're starting with one way referencing many nodes in a list.
The desired format is one node referencing multiple ways,
and then counting how many ways per node.

Also, we want to make sure that the starting list of node
references is unique.

First, unwind the node_refs list, then re-add them to a set
'unique_nodes'.  Then we can unwind the set so that
each document has one way and one node.

Then we can group by node and count the number of ways,
```

as well as add the id of each way to a set.
"""

```
query = [
    {"$match": {"type": "way"}},
    {"$unwind": "$node_refs"},
    {"$group": {"_id":{"way_id": "$id"},
                "unique_nodes": {"$addToSet": "$node_refs"}
               }},
    {"$unwind": "$unique_nodes"},
    {"$group": {"_id": {"node_id":"$unique_nodes"},
                "way_count": {"$sum": 1},
                "way_set": {"$addToSet": "$_id.way_id"}
               }},
    {"$sort": {"way_count": -1}}
    #,{"$limit":3}
]
node_l = [c['way_count'] for c in db.berkeley.aggregate(query)]

plt.hist(node_l);
plt.title("Most nodes are linked to just one way");
plt.xlabel("number of ways that refer to a particular node");
plt.ylabel("count of nodes");
```




```
query = [
    {"$match": {"type": "way"}},
```

```
    {"$unwind": "$node_refs"},
    {"$group": {"_id":{"way_id": "$id"},
                "unique_nodes": {"$addToSet": "$node_refs"}
               }},
    {"$unwind": "$unique_nodes"},
    {"$group": {"_id": {"node_id":"$unique_nodes"},
                "way_count": {"$sum": 1},
               }},
    {"$group": {"_id":"avg_way_count_per_node",
                "avg_way_count": {"$avg": "$way_count"}
               }},
    {"$limit":3}
    ]
node_l = [c for c in db.berkeley.aggregate(query)]
Node_l
============================================================
[{u'_id': u'avg_way_count_per_node', u'avg_way_count': 1.1447853309926048}]
============================================================
```

Additional Ideas

The key value format of tagging allows for flexibility, but also makes standardization more difficult. For instance, some tags say “car share” while others say “car sharing”, and there’s not necessarily a single correct answer. I think one way to help contributors support standardization is to run their text through a search of other tag keys, and to display similar words (as determined by a certain edit distance). The set of keys to search within could be defined as all tags that are in the same city, or all tags within a defined distance from the new data point.