RileyMShea/DataWranglingC

Notebook:	Cabinet
Created:	5/2/2019 5:01 PM
URL:	https://github.com/RileyMShea/DataWranglingC/blob/master/Final_Project/Final_submi...

OpenStreetMap Data Case Study

```
*Adapted from https://gist.github.com/carlward/54ec1c91b62a5f911c42#file-
```

Map Area

Rochester, NY United States

- https://www.openstreetmap.org/export#map=12/43.1721/-77.5521

Rochester, NY is my hometown so I thought it would be interesting to see if I could learn more about the surrounding area.

Problems Encountered in the Map:

After parsing the XML with BeautifulSoup and inserting the records into MongoDB, I noticed a couple of main problems with the data:

```
For some extra experience I opted to use a combination of BeautifulSoup a
```

- Non-standardized street suffixes
- Inconsistent postal codes *("1445033", "14607-2082", "14617-1822")*

Overabbreviated Street Names

Once the data was imported to MongoDB, some basic querying revealed street suffix and postal code inconsistencies. To deal with correcting street

suffixes I first grabbed a table with common street suffixes from the usps website with beautiful soup and pandas. This allowed me to programmatically clean the the suffixes to the USPS preferred format.

```
def get_suffix_mapping():
    """get a mapping for street suffixes by scraping table from USPS"""
    headers = {'User-agent': 'Mozilla/5.0'} # necessary to spoof user-age
    r = requests.get(r"https://pe.usps.com/text/pub28/28apc_002.htm", hea

    soup = BeautifulSoup(r.content, 'lxml')

    postal_table = soup.find('table', {'id':'ep533076'})

    df = pd.read_html(str(postal_table), header=0)[0]
    df.columns = ['Primary', 'Common', 'Standard']

    from collections import OrderedDict, defaultdict
    keys = df.iloc[:,1:3]

    return keys.to_dict(orient='records')
```

```
{'Common': 'ALLEE', 'Standard': 'ALY'},
{'Common': 'ALLEY', 'Standard': 'ALY'},
{'Common': 'ALLY', 'Standard': 'ALY'},
{'Common': 'ALY', 'Standard': 'ALY'},
{'Common': 'ANEX', 'Standard': 'ANX'}
```

With that dictionary I then iterated over the streets and performed an update_many on the MongoDB collection:

```
modified_count = 0
for entry in tqdm_notebook(distinct_streets):
    suffix = entry.split()[0]
    for mapping in mapping_dict:
        if mapping['Common'] == suffix.upper():
            #print(f"Changing {suffix} to {mapping['Standard']}")
            #print(entry.replace(suffix, mapping['Standard']))
            result = osm_col.update_many({'addr:street': entry}, {"$set"
            modified_count += result.modified_count
            continue
print(f"{modified_count} address suffixes updated")
```

```
for suffix in distinct_suffix:
    for mapping in mapping_dict:
        if mapping['Common'] == suffix.upper():
            print(f"Changing {suffix} to {mapping['Standard']}")
            continue
```

running db.rochester_osm.distinct("addr:street") in mongoshell we can now see the streets are updated to:

```
"East AVE",
"Paul RD",
"Kings Highwat S",
"Scottsville RD",
"Ridge Road East"...
```

As can be seen there is still some issues with the data, notably:

- Highwat should undoubtably be Highway

> Programatically cleaning each spelling error could prove to be difficult and time-intensive

- 'Road' in the last entry should be RD.

> Could deal with this by iterating over each word in the address instead of the just the last but that probably presents its own set of issues

For now we'll move on to cleaning some postal code data

Postal Codes

Fixing postal codes involved finding unique zip codes and auditing them to ensure they followed the same format, namely:

> 5 digits all numbers

Regardless, after standardizing inconsistent postal codes, some altogether “incorrect” (or perhaps misplaced?) postal codes surfaced when grouped together with this aggregation pipeline query:

```
unique_zip_codes = osm_col.distinct('addr:postcode')
pprint(unique_zip_codes)
```

```
update_dict = {'modified': 0,
               'deleted': 0,
               'good': 0}
for zip in tqdm_notebook(unique_zip_codes):
    if zip[0:5].isdigit() and len(zip) > 5:
        result = osm_col.update_many({'addr:postcode': zip}, {"$set": {'a
        update_dict['modified'] += result.modified_count
    elif not zip.isdigit() and len(zip)!=5:
        result = osm_col.delete_many({'addr:postcode': zip})
        update_dict['deleted'] += result.deleted_count
    elif zip.isdigit() and len(zip)==5:
        update_dict['good'] += 1
```

Here are the top ten results, after cleaning, beginning with the highest count:

```
pprint(list(osm_col.aggregate([
    {
        '$group': {
            '_id': '$addr:postcode',
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
        }
    }
]))[1:11])
```

```
[{'_id': None, 'count': 452824},
 {'_id': '14450', 'count': 1624},
 {'_id': '14624', 'count': 445},
 {'_id': '14618', 'count': 421},
 {'_id': '14623', 'count': 392},
 {'_id': '14534', 'count': 346},
 {'_id': '14626', 'count': 300},
 {'_id': '14514', 'count': 264},
 {'_id': '14612', 'count': 226},
 {'_id': '14620', 'count': 177}]
```

As we see here the majority of documents do not have a zip code listed in the 'addr:postcode' field. After Our cleaning though the zip codes that we do have all appear to be valid for the Rochester area and in the correct 5 digit format

Sort cities by count, descending

```
#City counts
city_counts = list(osm_col.aggregate([
    {
        '$group': {
            '_id': '$addr:city',
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
        }
    }
]))
```

And, the results, edited for readability:

```
None                452842
Rochester           2227
Fairport            1612
Pittsford           350
North Chili         264
Brighton            205
Greece              161
Webster             86
Henrietta           35
Churchville         29
West Henrietta      18
East Rochester      17
Rochester, NY       16
Penfield            16
Perinton            15
Hilton              15
Spencerport         11
North Greece        5
Macedon             5
Riga                2
rochester           2
Ontario, NY         2
Walworth            2
Gates               2
Irondequoit         2
Ontario             1
W Commercial St     1
Rochestet           1
```

```
pittsford           1
East Rochester Town 1
```

Here we see most of the entries are valid but some are:

- duplicated with differing capitalization
- formatted incorrectly(including the State)
- Typos, completely wrong field

This could be cleaned by:

- using the python string method `title()` on each value
- using the python string method `split(',')[0]` to trim off the entries that include the state
- The documents with mispellings would probably either need to be dropped or cleaned manually

similar to the zip_code cleaning you could them perform an update_many on the collection for the values that need to be fixed.

Data Overview and Additional Ideas

This section contains basic statistics about the dataset, the MongoDB queries used to gather them, and some additional ideas about the data in context.

File sizes

```
ls -sX -h
```

```
total 270M
   0 __pycache__             4.0K Untitled.ipynb   36K test_scientific_ex
1.0K street_affixes           76K bs42nd.ipynb     90M rochester_osm.jso
 16K Final_submission.ipynb  4.0K bs4stuff.ipynb   90M rochester_ny.osm
```

Number of nodes

```
db.rochester_osm.find({"type": "node"}).count()
```

> 405415

Number of ways

```
db.rochester_osm.find({"type": "way"}).count()
```

> 52523

Number of unique users

```
db.rochester_osm.distinct("user").length
```

719

Top 20 contributing users

```
db.rochester_osm.aggregate({
  '$sortByCount': "$user"
})
```

```
{ "_id" : "woodpeck_fixbot", "count" : 57093 }
{ "_id" : "wambag", "count" : 48318 }
{ "_id" : "dankpoet", "count" : 30128 }
{ "_id" : "idrive66", "count" : 29742 }
{ "_id" : "ECRock", "count" : 23697 }
{ "_id" : "sivart", "count" : 22811 }
{ "_id" : "RussNelson", "count" : 18800 }
{ "_id" : "timr", "count" : 18025 }
{ "_id" : "T-Rex", "count" : 15871 }
{ "_id" : "visionsofkenobi", "count" : 15249 }
{ "_id" : "URcommunications", "count" : 11601 }
{ "_id" : "stuuf", "count" : 11024 }
{ "_id" : "Craig Williams", "count" : 8023 }
{ "_id" : "paperboat", "count" : 6313 }
{ "_id" : "devrintalen", "count" : 5789 }
{ "_id" : "gadget", "count" : 5785 }
{ "_id" : "fx99", "count" : 5584 }
{ "_id" : "ColumM", "count" : 5127 }
{ "_id" : "bot-mode", "count" : 4931 }
{ "_id" : "Nathan Willard", "count" : 4491 }
```

Number of users appearing only once (having 1 post)

```
get_single_users(osm_col)
'146 users with only one post out of 719'
```

Additional Ideas

Contributor statistics

The contributions of users seems somewhat skewed, this seems like it could partly be due to some users being bots. Here are some user percentage statistics:

- Top user contribution percentage (“woodpeck_fixbot”) 12.46%
- Combined top 2 users' contribution (“wambag” and “woodpeck_fixbot”) 23.02%
- Combined top 10 users contribution 61.09%
- Combined number of users making up only 1% of posts 287 (about 24.90% of all users)

Additional Data Exploration

Top 10 appearing amenities

```
top_ten_amenities(col=osm_col)[0:10]
```

```
                _id  count
0              None  49953
1           parking   1828
2        restaurant    127
3            school     85
4              fuel     56
5         fast_food     54
6  place_of_worship     49
7              bank     46
8           shelter     43
9      fire_station     32
```

Places of worship by religion

```
religion = list(osm_col.aggregate([
    {
        '$match': {
            'amenity': {
                '$eq': 'place_of_worship'
            }
        }
    }, {
        '$group': {
            '_id': '$religion',
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
        }
    }
]))
```

```
{'_id': 'christian', 'count': 119},
 {'_id': None, 'count': 19},
 {'_id': 'jewish', 'count': 2},
 {'_id': 'buddhist', 'count': 1},
 {'_id': 'muslim', 'count': 1}
```

Most popular cuisines

```
# Most popular cuisine in restaurants
cuisine = list(osm_col.aggregate([
    {
        '$match': {
            'amenity': {
                '$eq': 'restaurant'
            }
        }
    }, {
        '$group': {
            '_id': '$cuisine',
            'count': {
                '$sum': 1
            }
        }
    }, {
        '$sort': {
            'count': -1
```

```
        }
    }
]))
pprint(cuisine[0:10])
```

```
[{'_id': None, 'count': 119},
 {'_id': 'pizza', 'count': 31},
 {'_id': 'american', 'count': 25},
 {'_id': 'italian', 'count': 15},
 {'_id': 'burger', 'count': 14},
 {'_id': 'sandwich', 'count': 8},
 {'_id': 'mexican', 'count': 8},
 {'_id': 'chinese', 'count': 7},
 {'_id': 'japanese', 'count': 5},
 {'_id': 'thai', 'count': 4}]
```

Conclusion

After this review of the data it's obvious that the Rochester area is incomplete, though significant progress has been made in cleaning it. It's clear that the OSM maps have large number of issues with data integrity where programatically cleaning the data cleaning the data after user submission may not be feasible and/or sufficient. Luckily I found that that openstreetmap has a number of resources available to help with quality assurance https://wiki.openstreetmap.org/wiki/Quality_assurance. My guess is with aid of these QA tools you could probably audit the data rather thoroughly though the completeness and timeliness of the data could still be an issue. In it's current state it seems okay for exploration, but for mission critical data it seems like it might be better to simply use a more established map service.