PROJECT 3: OPEN STREET MAP

Data Wrangling with MongoDB

Map Area: Chicago, IL, United States

Wesley Strange
ws2234@att.com

Contents

OSM File Used

Link to the Chicago, Illinois OSM XML data file:
https://s3.amazonaws.com/metro-extracts.mapzen.com/chicago_illinois.osm.bz2

The Chicago, Illinois dataset includes data for Chicago, Illinois and all surrounding suburban areas. I chose this area to audit because I live in one of the Northwest Suburbs and I thought this would be a good opportunity to learn about the surrounding areas.

Problems Encountered

Building

I wanted to combine all of the building fields into the same dictionary within node. I ran into an issue with the node[“building”] already being created as a string object when trying to assign a new field to the node[“building”] dictionary. TypeError: “'str' object does not support item assignment” was triggered. In order to fix this I had to catch the tag.attrib[“k”] equal to “building” before they were assigned as a string value and assign it as the “type” field in the node[“building”] dictionary.

```
    #elif no colon is found in tag.attrib["k"]
    elif tag.attrib["k"].find(":") == -1:
        if tag.attrib["k"] == "building":  # catch "building" tag
            if "building" not in node:     # up front and assign to the
                node["building"] = {}      # node["building"]["type"]
            node["building"]["type"] = tag.attrib["v"]  # field
        else:
            node[tag.attrib["k"]] = tag.attrib["v"]

# iterates through each nd in the element and stores them in list
for nd in element.iter("nd"):
    if "node_refs" not in node:
        node["node_refs"] = []  # adds "node_refs" list if absent
    node["node_refs"].append(nd.attrib["ref"])
```

Phone Numbers

A quick view of the dataset revealed a consistency problem with the node[“contact”][“phone”] field.

1. There were many different formats for the phone numbers found within the data. I wanted to remove the inconsistencies so that all of the phone numbers were in a consistent format of (123)456-7890. Examples of cleaned inconsistencies: {“+1 312 2634988”, “+1 773 462 9201”, “708-456-2000”, “(815) 725-8686”}.

```
def audit_phone_number(phone):
    phone = phone.translate(None, "+ -()")     #remove all non-digit chars

    # if phone number is 10 or 11 digits return reformatted phone number
    # else return None since the phone number is invalid
    if len(phone) == 10:
        phone = "("+phone[0:3]+")"+phone[3:6]+"-"+phone[6:]
        return phone
    elif len(phone) == 11:
        phone = "("+phone[1:4]+")"+phone[4:7]+"-"+phone[7:]
        return phone
    else:
        return None
```

Country

This wasn’t a problem, but I wrote this quick audit check to confirm the country was always returned as the standard two-letter “US” value. Just in case there was a country value that didn’t match this standard (ex. “USA”).

```
def audit_country(country):
    return "US"
```

State

A quick view of the dataset revealed a couple problems with the node[“address”][“state”] field.

1. There were records that were not from Illinois (ex: from the neighboring state of Indiana). Since this dataset is named “Chicago, IL,” I chose to discard all records that were not from Illinois.
2. There were some records where the node[“address”][“state”] field was equal to some form of Illinois (ex. “Il”, “illinois”, etc.). I wanted to retain these records, so I identified them by using the correct_state_list mapping list, then I updated the value to the correct “IL” value.

```
def audit_state(state, flag):
    if state == "IL":
        return state, flag # return as-is since it's equal to the standard "IL"
    elif state in correcting_state_list:
        return "IL", flag  # return state as the corrected standard "IL" value
    else:
        flag = True
        return state, flag # state is not Illinois, return flagged as bad data
```

Counties

There were several issues encountered when auditing the node[“address”][“county”] field.

1. Node[“address”][“county”] did not contain a valid county in Illinois. These records were discarded.

```
elif county:
    # option1: if county is valid, then return it
    # option2: return county flagged as bad data
    if county in county_postcode_mapping:
        return county, flag  # return valid county
    else:
        return county, True  # return flagged as bad data
```

2. Node[“address”][“county”] field was absent, but there was a node[“tiger”][“county”] field that contained a value for county. For these records, I attempted to fill in the missing node[“address”][“county”] data by pulling it from the node[“tiger”][“county”] field. A second issue was realized during this process, since the node[“tiger”][“county”] data was not in a consistent format that could immediately be used. Therefore, I had to do some manipulations to drill down to the actual county value. Example of inconsistent data: {"Lake, IL", "IL; Lake", "IL: Lake"}.

```
elif tiger_county:
    # tiger_county contains county and state in some order
    # examples: ("Cook, IL" or "IL;Cook")
    # this line splits the county and state into two separate variables
    tiger_county = re.split("[,;:]", tiger_county)

    # option1: if tiger_county list contains a valid county, then return it
    # option2: return county flagged as bad data
    if type(tiger_county) is list and len(tiger_county) >= 2:
        tiger_county[0] = tiger_county[0].strip()
        tiger_county[1] = tiger_county[1].strip()
        if tiger_county[0] == "IL" or tiger_county[1] == "IL":
            if tiger_county[0] in county_postcode_mapping:
                return tiger_county[0], flag  # return valid tiger_county
            elif tiger_county[1] in county_postcode_mapping:
                return tiger_county[1], flag  # return valid tiger_county
            else:
                return tiger_county, True  # return as bad data
        else:
            return tiger_county, True  # return as bad data
    else:
        return county, True  # return as bad data
```

3. County was invalid for both node[“address”] and node[“tiger”], so the records were discarded as bad data. Or county was invalid for node[“address”], but valid for node[“tiger”], so the node[“tiger”] valud was returned.

```
if county and tiger_county:
    # tiger_county contains county and state in some order
    # examples: ("Cook, IL" or "IL;Cook")
    # this line splits the county and state into two separate variables
    tiger_county = re.split("[,;:]", tiger_county)

    # option1: if county is valid, then return it
    # option2: if tiger_county list contains a valid county, then return it
    # option3: return county flagged as bad data
    if county in county_postcode_mapping:
        return county, flag  # return valid county
    elif type(tiger_county) is list and len(tiger_county) >= 2:
        tiger_county[0] = tiger_county[0].strip()
        tiger_county[1] = tiger_county[1].strip()
        # first checks to make sure the county is in Illinois
        if tiger_county[0] == "IL" or tiger_county[1] == "IL":
            if tiger_county[0] in county_postcode_mapping:
                return tiger_county[0], flag  # return valid tiger_county
            elif tiger_county[1] in county_postcode_mapping:
                return tiger_county[1], flag  # return valid tiger_county
            else:
                return tiger_county, True  # return as bad data
        else:
            return county, True  # return as bad data
    else:
        return county, True  # return as bad data
```

Postal Codes

There were several issues encountered when auditing the node[“address”][“postcode”] field.

1. Inconsistencies found in the format used for the node[“address”][“postcode”] and node[“tiger”][“zip-left”] fields. The postcode was updated to use a standard five-digit format of 12345. Ex. of inconsistent format: “60007-3445”.

```
# trims the postcode down to a 5 digit integer value for consistency
# returns None if error is encountered
try:
    postcode = int(postcode[0:5])
except (TypeError, ValueError):
    postcode = None
# trims the tiger_postcode down to a 5 digit integer value for consistency
# returns None if error is encountered
try:
    tiger_postcode = int(tiger_postcode[0:5])
except (TypeError, ValueError):
    tiger_postcode = None
```

2. Node[“address”][“postcode”] did not contain a valid Illinois postcode.

```
elif postcode:
    if postcode in postcode_county_mapping:
        return postcode, flag  # return valid illinois postcode
    else:
        return postcode, True  # return flagged as bad data
```

3. Node[“address”][“postcode”] field was absent, but there was a node[“tiger”][“zip-left”] field that contained a value for postcode. For these records, I attempted to fill in the missing node[“address”][“postcode”] data by pulling it from the node[“tiger”][“zip-left”] field.

```
elif tiger_postcode:
    if tiger_postcode in postcode_county_mapping:
        return tiger_postcode, flag  # return valid illinois postcode
    else:
        return postcode, True  # return flagged as bad data
```

4. Postcode was invalid for both node[“address”] and node[“tiger”], so the records were discarded as bad data. Or postcode was invalid for node[“address”], but valid for node[“tiger”], so the node[“tiger”] value was returned.

```
if postcode and tiger_postcode:
    if postcode in postcode_county_mapping:
        return postcode, flag   # return valid illinois postcode
    elif tiger_postcode in postcode_county_mapping:
        return tiger_postcode, flag  # return valid illinois postcode
    else:
        return postcode, True  # return flagged as bad data
```

Cities

A quick view of the dataset revealed a couple problems with the node[“address”][“city”] field.

1. There were records that were not from Illinois (ex: from the neighboring state of Indiana). Since this dataset is named “Chicago, IL,” I chose to discard all records that were not from Illinois.
2. There were some records where the node[“address”][“city”] field was equal to some form of a valid Illinois city (ex. {'Woodbridge':'Woodridge','Carpentesville':'Carpentersville’}). I wanted to retain these records, so I identified them by using the correcting_cities_mapping dictionary, then I updated the value to the correct city name.

```
def audit_city(city_name,flag):
    if city_name in cities_in_illinois:
        return city_name, flag  # return as-is since it's a valid city
    elif city_name in correcting_cities_mapping:
        return correcting_cities_mapping[city_name], flag  # return corrected
    else:                                                  # city name
        flag = True
        return city_name, flag  # city is not valid, return flagged as bad data
```

Street Names

A quick view of the dataset revealed a couple problems with the node[“address”][“street”] field.

1. There were a few instances where the street name had to be manually corrected due to unique formatting. I identified them by using the correcting_cities_mapping dictionary, then I updated the value to the correct street name. Example: {"1050 Essington Rd. Joliet, IL 60435":"Essington Road"}.

```
# Manually fixes some of the incorrect street nmaes using mapping dict
if street in street_names_mapping:
    street = street_names_mapping[street]
```

2. There were a few instances that had a house number listed at the beginning of the street name. This was inconsistent with the majority of the dataset, so I removed the house number. Example: {‘100 Grant Road’}.

```
housenumber = housenumber_re.search(street)
# if housenumber is found in street field it is removed.
if housenumber:
    housenumber = housenumber.group()
    street = street.replace(housenumber,"")  # removes housenumber
    street = street.strip()  # strip leading/trailing blanks
```

3. There were many instances where the street name was over abbreviated. These inconsistencies in the street prefixes and suffixes were fixed by using dictionary lookup mappings. Examples of inconsistencies: {'W Lincoln HWY', 'N. Golf Rd.'}.

```
# audit the street field to fix prefix/suffix inconsistencies
i = 0
s = street.split(" ")    # split field into multiple words
updated_street = ""
while i < len(s):
    if s[i] in street_suffixes_mapping:
        s[i] = street_suffixes_mapping[s[i]]  # cleans suffix
    elif s[i] in street_prefixes_mapping:
        s[i] = street_prefixes_mapping[s[i]]  # cleans prefix
    updated_street = updated_street + s[i] + " "    # recombine words
    i = i + 1
street = updated_street.strip()
```

4. There were many inconsistencies found in the name formatting used for the highway/route names. I wrote code to clean up these inconsistencies and make the highway/route a consistent format. Examples of inconsistencies: {'IL 59', 'Route 59', 'Route IL 59', 'S Rte 59', 'U.S. 14'}.

```
# audit the street field to fix inconsistencies in the naming
# convention of the IL/US highways. This is completed by replacing
# common inconsistencies with consistent name. ("Route 59" --> "IL-59".
if street.find("Route") >= 0 or street.find("US") >= 0:
    if street.find("IL Route ") >= 0:
        street = street.replace("IL Route ","IL-")
    elif street.find("Route IL ") >= 0:
        street = street.replace("Route IL ","IL-")
    elif street.find("Route ") >= 0:
        street = street.replace("Route ","IL-")
    elif street.find("US Highway ") >= 0:
        street = street.replace("US Highway ","US-")
    elif street.find("US ") >= 0:
        street = street.replace("US ","US-")
    elif street.find("U.S. ") >= 0:
        street = street.replace("U.S. ","US-")
```

Full Address

A quick view of the dataset revealed a couple problems with the node["address"]["full"] field.

1. The same audits that were performed on the node["address"]["street"] field were also performed on the node["address"]["full"] field.
2. Plus, one additional audit was performed to ensure that the node["address"]["full"] value matched the combined node["address"]["housenumber"] and node["address"]["street"] fields. If they didn't match, then the more prominent value out of the two was returned for the node["address"]["full"] field.

```
# validate that the combination of node["address"]["housenumber"] and
# node["address"]["street"] is the same value as node["address"]["full"]
# if not, then return the more prominent value out of the two
if "street" in node["address"] and "housenumber" in node["address"]:
    address = node["address"]["housenumber"] + " " + \
                                              node["address"]["street"]
    if node["address"]["full"].find(address) >= 0:
        return node
    else:
        if len(address) > len(node["address"]["full"]):
            node["address"]["full"] = address
        return node
```

Address Completion

There were many records in the dataset where the address fields were not as “complete” as I thought they should be. For example, there were situations where the user only entered the Postal Code, but not the County, State, or Country – all of which can be derived from the provided Postal Code. Therefore, I wrote a function to make the address more complete.

```
def audit_address_completeness(node):
    if "postcode" in node["address"]:
        if "county" not in node["address"]:
            node["address"]["county"] = postcode_county_mapping[
                                                node["address"]["postcode"]]
        if "state" not in node["address"]:
            node["address"]["state"] = "IL"
        if "country" not in node["address"]:
            node["address"]["country"] = "US"
    elif "city" in node["address"]:
        if "state" not in node["address"]:
            node["address"]["state"] = "IL"
        if "country" not in node["address"]:
            node["address"]["country"] = "US"
    elif "county" in node["address"]:
        if "state" not in node["address"]:
            node["address"]["state"] = "IL"
        if "country" not in node["address"]:
            node["address"]["country"] = "US"
    elif "state" in node["address"]:
        if "country" not in node["address"]:
            node["address"]["country"] = "US"
    return node
```

Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File Sizes

chicago_illinois.osm: 1,915,347 KB
chicago_illinois.osm.json: 2,112,859 KB

Number of Documents

```
> db.chicagoIL.find().count()
```

9362876

Number of Nodes

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

8222625

Number of Ways

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

1140056

Top Users

```
> db.chicagoIL.aggregate([{"$group": {"_id": "$created.user", "count": {"$sum": 1}}},{"$sort": {"count": -1}},{"$limit": 10}])
```

[{u'count': 5645878, u'_id': u'chicago-buildings'}, {u'count': 1108451, u'_id': u'Umbugbene'}, {u'count': 238319, u'_id': u'alexrudd (NHD)'}, {u'count': 230114, u'_id': u'woodpeck_fixbot'}, {u'count': 110625, u'_id': u'patester24'}, {u'count': 106853, u'_id': u'TIGERcnl'}, {u'count': 105329, u'_id': u'mpinnau'}, {u'count': 90488, u'_id': u'asdf1234'}, {u'count': 85512, u'_id': u'Sundance'}, {u'count': 83894, u'_id': u'g246020'}]

Top Amenities:

```
> db.chicagoIL.aggregate([{"$match": {"amenity": {"$exists": 1}}},{"$group": {"_id": "$amenity", "count": {"$sum": 1}}},{"$sort": {"count": -1}},{"$limit": 10}])
```

[{u'count': 11589, u'_id': u'parking'}, {u'count': 4279, u'_id': u'place_of_worship'}, {u'count': 3428, u'_id': u'school'}, {u'count': 1773, u'_id': u'restaurant'}, {u'count': 1188, u'_id': u'fast_food'}, {u'count': 718, u'_id': u'fuel'}, {u'count': 502, u'_id': u'bank'}, {u'count': 456, u'_id': u'grave_yard'}, {u'count': 402, u'_id': u'cafe'}, {u'count': 320, u'_id': u'bicycle_rental'}]

Top Religions:

```
> db.chicagoIL.aggregate({"$match": {"amenity": {"$exists":1}, "amenity": "place_of_worship"}}, {"$group": {"_id": "$religion", "count": {"$sum": 1}}},{"$sort": {"count":-1}}, {"$limit": 10}])
```

[{u'count': 4036, u'_id': u'christian'}, {u'count': 118, u'_id': None}, {u'count': 86, u'_id': u'jewish'}, {u'count': 17, u'_id': u'muslim'}, {u'count': 11, u'_id': u'buddhist'}, {u'count': 2, u'_id': u'hindu'}, {u'count':

2, u'_id': u'unitarian_universalist'}, {u'count': 1, u'_id': u'*'}, {u'count': 1, u'_id': u'sikh'}, {u'count': 1, u'_id': u'ascended_master_teachings'}]

Top Cuisines:

> db.chicagoIL.aggregate([{"$match": {"amenity": {"$exists":1}, "amenity": "restaurant"}},{"$group": {"_id": "$cuisine", "count": {"$sum": 1}}},{"$sort": {"count":-1}},{"$limit": 10}])

[{u'count': 771, u'_id': None}, {u'count': 143, u'_id': u'pizza'}, {u'count': 127, u'_id': u'mexican'}, {u'count': 83, u'_id': u'american'}, {u'count': 76, u'_id': u'italian'}, {u'count': 57, u'_id': u'chinese'}, {u'count': 57, u'_id': u'burger'}, {u'count': 39, u'_id': u'sandwich'}, {u'count': 35, u'_id': u'thai'}, {u'count': 24, u'_id': u'japanese'}]

Other Ideas About Dataset

Additional Auditing

County

For the county audit, I validated that the county name is a valid County in Illinois, however, the county could still be invalid if the neighboring state Indiana had a county with the same name. I could spend more time thoroughly auditing the county field, but I think the current level of auditing is sufficient for this project.

There is also a field called “is-in” that contains a value which appears to be county (ex. “Kane, IL”). This “is-in” field could be used perform additional validations on the node[“address”][“county”] field. Possible validations include:

- Cross-validate that the county in node[“is-in”] and node[“address”][“county”] are the same.
- Drive the node[“address”][“county”] from the node[“is-in”] field.

Distinct Users

There were User Name’s that included problem characters (e.g. “/” or “\”) that prevented me from running a query to get a list of all of the user’s who contributed data in the dataset. It would take further investigation on my part to learn how to complete this query.

db.chicago.distinct({"created.user"}).length

Msg: "instance of %s" % (string_type.__name__,)) TypeError: key must be an instance of basestring

Postcode

Postcode returned as bad data, but valid data might have been contained at the end. I should have removed all of the non-digit characters before trimming it down to five digits. Examples of data that was discarded, but had valid date inside the field.

- ‘Wasco, IL 60183’
- ‘IL, 60642’

Non-Illinois Data

I identified possible non-Illinois data that made it through the cleaning process because there was no node[“address”] field to audit. If I had additional time I would write audits to clean the tiger geolocation fields to remove the non-Illinois data.

Possible Benefits

The benefit of removing the non-Illinois data is that the dataset will then only contain Chicago and Chicago Suburban data which will more closely match the dataset name “Chicago, Illinois”.

Possible Problems

A possible problem that could be encountered is that the TIGER data being used to filter out the non-Illinois data may not be 100% accurate. For example, I learned on the OSM wiki website that the TIGER data was originally created for taking census surveys, so the roads may not be aligned to their true position, but close enough for the census workers to find the location of the houses. Therefore, there could be some instances along the Illinois-Indiana border where the

TIGER data shows “Indiana” as the state but the data is actually falls in “Illinois” or vice versa. This could lead to incorrect cleaning of the data.

References

- http://www.unitedstateszipcodes.org/il/
- https://en.wikipedia.org/wiki/List_of_towns_and_villages_in_Illinois
- https://wiki.openstreetmap.org/wiki/Map_Features
- https://www.python.org/dev/peps/pep-0008/
- http://www.hongkiat.com/blog/source-code-comment-styling-tips/
- https://discussions.udacity.com/t/how-to-use-mongodb-locally/13420/15
- https://docs.python.org/2/howto/regex.html
- http://wiki.openstreetmap.org/wiki/TIGER_fixup