DataAnalystNanoDegreeOpenStreetMap

Author: James Tench **Project**: Udacity Data Analyst project 3

Project Summary

This project is the final project associated with the Data Wrangling course. The purpose of the course, and project is to demonstrate the process of converting raw xml to a usable format for data analysis.

The conversion process involved auditing, cleaning, and shaping the data into a usable schema in json format. Python was used for pre-processing the data

MongoDb or SQL were both options for completing the project. MongoDB was selected for this project.

Data source: Mapzen Map area: Cleveland Ohio File Size: 362.4 MB uncompressed

---

Project Criteria

1. Problems encountered with the map

Inconsistent Street names

Inconsistent street names were found throughout the file. Most common cases were abbreviations for Street with St. or Road rd. To correct the names based on the common naming in the area the python code was updated to map and update the values during the shaping process. This was done with a python dictionary. `street_name_mapping = { 'St': 'Street', 'Ave': 'Avenue', 'Ave.': 'Avenue', 'Blvd': 'Boulevard', 'Blvd.': 'Boulevard', 'Dr': 'Drive', 'Dr.': 'Drive', 'Ln': 'Lane', 'Pkwy': 'Parkway', 'Rd': 'Road', 'Rd.': 'Road', 'St': 'Street', 'St.': 'Street', 'Street.': 'Street', 'ave': 'Avenue' }`

Inconsistent characters

During the data query phase, while running aggregate queries in MongoDB these inconsistencies were found. Specifically the single quote character was represented with the unicode \u2019 and the "'" symbol. This was causing data to group in different ways.

At this point I went back to the data cleaning process and wrote a simple regular expression to change to one consistent character when the unicode character was encountered.

```
right_apos = re.compile(ur'\u2019', re.IGNORECASE)
matched = right_apos.search(value)
if matched:`
    value = re.sub(ur'\u2019', "'", value)
```

Problem keys in tags

My schema identified an entry by the type based openstreetmap model of nodes, ways and references. To account for this my schema has a field "type" which indicated what type of node. After the parsing process was complete I found other values for "type" than what was expected. To investigate the issue I added code to track all of the key names and the number of times they appeared. I found that there was a key on the "tag" nodes of "type". This key was overwriting the original expected value. Overall there were only 42 occurrences of this issue. To correct the problem I chose to ignore this tag as I was shaping the data. To control that key value and other keys that I knew were part of my schema I created an ignore list. `keys_to_ignore = ["type", "id", "visible", "created", "address"]`

Additional tags

During the tag counting process I found the following tags: `{'bounds': 1, 'member': 19661, 'nd': 1871102, 'node': 1610592, 'osm': 1, 'relation': 2548, 'tag': 1069095, 'way': 166319}`

I expected now, way, relation, tag and nd. The other tags were not expected. I could not find a need to include any of these tags in the data so the python code ignored those tags.

Inconsistent zip codes

Inspection of the zip code showed inconsistencies. In some cases, the zip code contained a "-" character. In other cases the zip code was the word "Ohio". I cleaned this data by only keeping the first 5 digits, and ignoring the "Ohio" tags. See shared_code.py and audit_street_names.py for implementation._

Example of inconsistent zip codes

```
    '44113',
    '44113-2960',
```

After cleaning the zip code data, I reloaded the MongoDB database and checked for the address fields that contained zip codes to confirm only 5 digit zip codes were included.

Missing zip code information

To allow more searches by zip code I saw there were keys for "tiger:zip_direction" in the xml data. After running a query there were over 47800 nodes that had tiger:zip_left alone that did not have any other postal code information. To clean the data and make it more usable I added this to my code when a postal code was not present and created an address field based on the tiger fields. I also cleaned those values before adding them. Additional cleaning included finding zip codes that were not in the range of valid Ohio zip codes such as 99999.

Unnecessary or unused tags

Aside from the address fields, I could not find a use for the other tiger data, so I decided to parse out the remaining tiger data.

2. Overview of the data

File size: cleveland_ohio.osm 362.4 MB Unique user count: 977 Number of nodes: 1610592 Number of ways: 166319

```
def number_of_nodes_by_type(type):
    db = get_db()
    return db.clevelandohio.find({"type": type}).count()

if True:`
    print "nodes: {0}".format(number_of_nodes_by_type("node"))
    print "ways: {0}".format(number_of_nodes_by_type("way"))
```

Number of chosen types of nodes (food locations)

```
def get_food_node_count(db):
    return db.clevelandohio.aggregate(
        [
            {
                "$match": {
                    "$or": [{"cuisine": {"$exists": True}},
                            {"$and": [{"amenity": {"$exists": True}}, {"amenity":
"restaurant"}]},
                            {"food": {"$exists": True}}
                            ]
                }
            },
            {"$group": {"_id": "food_total", "count": {"$sum": 1}}}
        ]
    )

if True:
    db = get_db()
    cursor = get_food_node_count(db)
    for pointer in cursor:
        print pointer
```

```
{u'count': 311, u'_id': u'food_total'}
```

Food locations

```
def get_restaurant_count(db):
    return db.clevelandohio.aggregate(
        [
            {
                "$match": {
```

```
                    "$or": [{"cuisine": {"$exists": True}},
                            {"$and": [{"amenity": {"$exists": True}}, {"amenity":
"restaurant"}]},
                            {"food": {"$exists": True}}
                            ]
                }
            },
            {
                "$project": {
                    "name": {"$toLower": "$name"}
                }
            },

            {
                "$group": {"_id": "$name", "count": {"$sum": 1}}
            },
            {
                "$sort": {"count": -1}
            }
        ]
    )

if True:
    db = get_db()
    cursor = get_restaurant_count(db)
    pretty_print_cursor(cursor)
```

Top 15 results `{u'_id': u"mcdonald's", u'count': 28}` `{u'_id': u'burger king', u'count': 15}` `{u'_id': u'bob evans', u'count': 13}` `{u'_id': u'pizza hut', u'count': 12}` `{u'_id': u"wendy's", u'count': 12}` `{u'_id': u'', u'count': 10}` `{u'_id': u"applebee's", u'count': 8}` `{u'_id': u'taco bell', u'count': 6}` `{u'_id': u"arby's", u'count': 5}` `{u'_id': u'dairy queen', u'count': 4}` `{u'_id': u'longhorn steakhouse', u'count': 4}` `{u'_id': u'olive garden', u'count': 3}` `{u'_id': u'starbucks', u'count': 3}` `{u'_id': u'subway', u'count': 3}` `{u'_id': u'ihop', u'count': 3}`

Top contributers

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

```
{u'_id': u'woodpeck_fixbot', u'count': 638581}
{u'_id': u'unigami', u'count': 104489}
{u'_id': u'skorasaurus', u'count': 78696}
{u'_id': u'Evan Edwards', u'count': 60062}
{u'_id': u'wlgann', u'count': 55629}
{u'_id': u'Johnny Mapperseed', u'count': 52250}
{u'_id': u'texnofobix', u'count': 48977}
{u'_id': u'Rub21', u'count': 39766}
{u'_id': u'bgarman4', u'count': 38855}
{u'_id': u'Bored', u'count': 37926}
```

Top users that documented food location

```
def get_user_food_contributions(db):
    return db.clevelandohio.aggregate(
        [
            {
                "$match": {
                    "$or": [{"cuisine": {"$exists": True}},
                            {"$and": [{"amenity": {"$exists": True}}, {"amenity":
"restaurant"}]},
                            {"food": {"$exists": True}}
                            ]
                }
            },
            {
                "$group": {"_id": "$created.user", "count": {"$sum": 1}}
            },
            {
                "$sort": {"count" : -1}
            },
            {
                "$limit": 10
            }
        ]
    )
```

```
{u'_id': u'skorasaurus', u'count': 66}
{u'_id': u'Evan Edwards', u'count': 37}
{u'_id': u'tmb926', u'count': 30}
{u'_id': u'SomeoneElse_Revert', u'count': 20}
{u'_id': u'Johnny Mapperseed', u'count': 18}
{u'_id': u'unigami', u'count': 12}
{u'_id': u'Minh Nguyen', u'count': 12}
{u'_id': u'kuduboet', u'count': 11}
{u'_id': u'texnofobix', u'count': 8}
{u'_id': u'kisaa', u'count': 8}
```

2. Other ideas

The basis for most of my queries was to explore some additional information about the data outside of simply cleaning and loading it into the database. I found it surprising the low number of

food nodes. What specifically drew my attention was that only 3 subway locations were reported. My knowledge of the area tells me there are many more locations.

This tells me that the data is not up to date, and this type of data analysis around the food industry in Cleveland would need an additional data source to supplement the database. This would require additional work to combine the data sets into one schema, but it could be done in a similar manner as this data was cleaned. Multiple JSON files could be created to load the data from various sources.

Pulling additional data would require some work to check for duplicate entries. If another source like google maps was used, one could attempt to match up latitude and longitude coordinates to combine data.

3. Conclusion

Overall the data analysis was minimal. The cleaning and processing was the majority of the work. This matched the general purpose of the project to "wrangle" data with python and load it into a data source. I found that moving data from dictionary structures into MongoDB is a good workflow and makes the data cleaning process much easier. My audit_street_names.py file does most of the cleaning. I added some helper functions to clean the data. I found issues with unicode characters and corrected for those. I also found issues with tags having keys that were equal to my high level schema keys so I created an ignore list.

Files 1. data_overview.py - some python code to inspect the raw data 2. audit_street_names.py - primary wrangling code. functions to clean and shape data 3. mongo_queries.py - queries to explore the data that was loaded to MongoDB