PROJECT REPORT

UDACITY

Contents

1. Code Functionality

All required Lesson 6 problems are correctly solved to the best of my ability with the submitted code. Python files for the code in lesson 6 are also attached in the zip file.

2. Code Readability

I have tried to write the code so that it follows an intuitive, easy-to-follow logical structure. Where I thought the code would have been difficult to understand, I have tried to add comments for easy readability.

NOTE:

Below were the steps taken in order to clean and organize the map data osm file before finally loading it into MongoDB:

1. Map file was exported from Openstreetmap and saved as minneapolis.osm (details in the “Project Summary.docx” file).

2. Modified version of the audit.py program (lesson 6) was used to identify problems in the map data (code excerpts are included in Section 3).

3. In addition to that, the original downloaded minneapolis.osm file was converted to .json format using the original data.py program (lesson 6) and passed onto MongoDB using the following command:

   ```
   >>>mongoimport --db test --collection project minneapolis.osm.json
   ```

   Some basic queries were applied to identify additional problems/inconsistencies in the data (details covered in section 3 below).

4. Then using a modified version of the data.py python program (modified_data.py – attached in the zip file) minneapolis.osm file was cleaned & organized and converted to final_minneapolis.osm.json

5. This final_minneapolis.osm.json file was imported into MongoDB using the following command:

   ```
   >>>mongoimport --db test --collection final_project final_minneapolis.osm.json
   ```

6. Finally, MongoDB queries were run using pymongo to answer questions in the “Section 4 - Overview of the data” section.

7. All the details of problems and inconsistencies are covered below in section 3 “Problems Encountered in the Map”

3. Problems Encountered in the Map

There were 2 methods used to identify possible problems and inconsistencies in the data.

Initially, audit.py (lesson 6) program with slight modifications in the **is_street_name ()** method was used to identify possible problems in the minneapolis.osm file.

Secondly, using the data.py (lesson 6) program, original minneapolis.osm was converted to minneapolis.osm.json and passed onto a MongoDB collection and queries were run via pymongo to identify additional issues in the map.

The main problems/inconsistencies observed were as follows:

1. Ambiguous/over-abbreviated street names
2. Inconsistent street direction conventions
3. Incorrect zip codes
4. Incorrect house number (1 instance)
5. Incorrect GaWC [Globalization and World Cities] ranking for Minneapolis (1 instance)
6. No units were seen for ele (elevation)

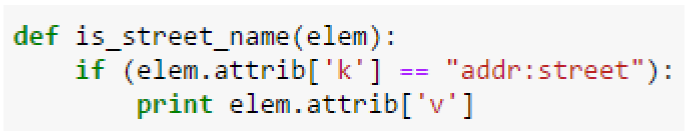
Problems and inconsistencies in the map data were resolved by modifying the data.py program. The update_name () method from audit.py was modified and also included in data.py to address the issue of street names.

NOTE: A test file example_test.osm was created and used to test different scenarios before applying the modified version of the data.py program i.e. modified_data.py to the minneapolis.osm file (example_test.osm is included in the zip file for reference).

Details of the problems/inconsistencies are as follows:

Ambiguous/Over-abbreviated Street Names and Inconsistent Street Direction Conventions

The original version of the audit.py program with slight modification in is_street_name () method was applied to the original downloaded map file minneapolis.osm to view the inconsistencies in abbreviated street names.



```
def is_street_name(elem):
    if (elem.attrib['k'] == "addr:street"):
        print elem.attrib['v']
```

MS excel, workbook ‘Full Street Names Minneapolis.xlsx’, was used for easy identification of problematic street names and is included in the zip file for reference. Here are some of the street names that seemed to be over-abbreviated:

- SE University Ave
- S Riverside Ave
- Fillmore St. NE
- SE Ontario St
- 10th Avenue SE & 5th Street SE

Some of the inconsistent street direction conventions are as follows:

- Dr, Drive
- Ln, Lane
- Ave, Ave., Avenue
- Blvd, Boulevard
- Pkwy, Parkway
- Rd, Road
- Pl, Plaza
- St, St., Street

The data.py program was modified to include the update_name () [from audit.py program] with slight changes to incorporate all the ambiguous and inconsistent street name scenarios (modified version of the data.py is in the zip file for reference).

Implementation of update_name () in data.py program stored the updated street names directly into a .json dictionaries as the output i.e.

minneapolis.osm → modified_data.py [including update_name ( )] → final_minneapolis.osm.json

This change addressed the ambiguities and updated the street names as follows:

- SE University Ave → Southeast University Avenue
- S Riverside Ave → South Riverside Avenue
- Fillmore St. NE → Fillmore Street Northeast
- SE Ontario St → Southeast Ontario Street
- 10th Avenue SE & 5th Street SE → 10th Avenue Southeast and 5th Street Southeast

Exploring the street names revealed the following problems/inconsistencies:

1. In one instance, zip code was accidently stored as street name. A slight modification in data.py program helped resolve the issue.

   ```
   elif second_k_value == "addr:street" and second_v_value == "55414":
       continue
   ```

2. In one case, Rd/Pkwy was used as an abbreviation. This was converted to Parkway as it was more frequently used throughout the file. (East River Rd/Pkwy converted to East River Parkway for consistency).

```
# mapping_last used from imroving last part of street address
mapping_last = {"St": "Street","St.": "Street", "Ave": "Avenue", "Rd.": "Road", "Pl" : "Plaza", "Blvd" : "Boulevard", "Dr" : "Drive",
                "SE" : "Southeast", "Ln" : "Lane", "Ave.": "Avenue", "N" : "North", "N." : "North", "NE" : "Northeast",
                "Pkwy" : "Parkway", "Rd": "Road", "Rd/Pkwy" : "Parkway", "S" : "South", "SE" : "Southeast", "W" : "West",
                "S.E." : "Southeast"}
```

3. In one case, street address was 10th Avenue SE & 5th Street SE and was converted to 10th Avenue Southeast and 5th Street Southeast.

```
if len(name_value) == 7:# correcting 7 words street names (only one instance: "10th Avenue SE & 5th Street Southeast")

    # correcting the street name
    name_value[2] = "Southeast"
    name_value[3] = "and"
    name = " ".join(name_value)
```

4. In another case, street address was 1320 4th St SE and was converted to 1320 4th Street Southeast.

```
if len(name_value) == 4:# correcting 4 words street names (only one instance: "1320 4th St SE")

    # correcting the street name
    name_value[2] = "Street"
    name = " ".join(name_value)
```

Incorrect Zip Codes

To check for the inconsistencies in zip codes, original file Minneapolis.osm was passed onto original data.py program and the resulting minneapolis.osm.json file was passed onto MongoDB and following command was run:

```
mongodb_project = db.project.aggregate([{"$match":{"address.postcode":{"$exists":1}}},
                                        {"$group":{"_id":"$address.postcode", "count":{"$sum":1}}}, {"$sort":{"count":-1}}])

pprint.pprint(mongodb_project)
```

This revealed 4 instances where the zip codes were incorrect

- `{u'_id': u'MN', u'count': 1}`
- `{u'_id': u'100', u'count': 1}`
- `{u'_id': u'Pillsbury Dr', u'count': 1}`
- `{u'_id': u'5114', u'count': 1}`

These inconsistencies were also identified by applying the following piece of code in the audit.py program to the minneapolis.osm file.

```
def is_street_name(elem):
    if (elem.attrib['k'] == "addr:postcode"):
        print elem.attrib['v']
```

Issues mentioned above were resolved by adding an elif conditions in the shape_eleme nt () method of data.py program.

```
elif second_k_value == "addr:postcode" and second_v_value == "MN": # addressing inconsistency in zip code
    node['address'][second_k_value[5:]] = "55408"
elif second_k_value == "addr:postcode" and second_v_value == "100": # addressing inconsistency in zip code
    node['address'][second_k_value[5:]] = "55401"
elif second_k_value == "addr:postcode" and second_v_value == "5114": # addressing inconsistency in zip code
    node['address'][second_k_value[5:]] = "55114"
elif second_k_value == "addr:postcode" and second_v_value == "Pillsbury Dr": # addressing inconsistency in zip code
    node['address'][second_k_value[5:]] = "55455"
```

Link below show the list of legal zip codes used in Minneapolis:
http://www.city-data.com/zipmaps/Minneapolis-Minnesota.html

Incorrect House Number

The only instance for incorrect house number was addressed with the following piece of code:

```
elif second_k_value == "addr:housenumber" and second_v_value == "2115 Summit Ave, Saint Paul, MN": # housenumber incorrect
    node['address'][second_k_value[5:]] = "2115"
    node['address']['street'] = 'Summit Avenue'
    node['address']['city'] = 'Saint Paul, MN'
```

No units for ele (elevation)

Following piece of code was included in data.py program to append units for elevation. In this case it was m (meters).

```
elif second_k_value.startswith('ele'):
    node['elevation'] = second_v_value + " m"
```

GaWC [Globalization and World Cities] ranking for Minneapolis

There was only one instance where GaWC ranking was included but was not updated. It showed Minneapolis to be in the ‘Gamma+’ ranking while the link http://en.wikipedia.org/wiki/Global_city revealed it to be in “Beta-“ ranking. This was updated via the following piece of code:

```
# updating and storing GaWC ranking
elif second_k_value.startswith('GaWC'):
    node['GaWC'] = "Beta-"
```

4. Overview of the Data

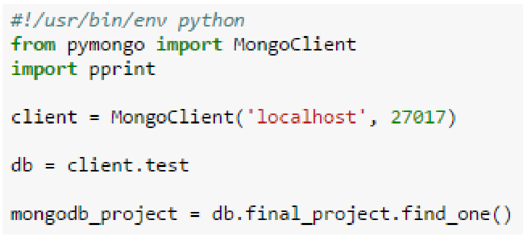
The modified_data.py program was used to clean and organize minneapolis.osm file by addressing the issues described in section 3 and convert it to final_minneapolis.osm.json.

This final_minneapolis.osm.json was then passed onto MongoDB using the following command:

>>>mongoimport --db test --collection final_project final_minneapolis.osm.json

Following the storage of the json document into MongoDB, various commands were executed using pymongo to address the questions asked in the rubric.

Ipython notebook was used for running the commands and initiation of the database was done as follows (mongodb_queries.py includes all the queries applied [attached in zip file]):



```python
#!/usr/bin/env python
from pymongo import MongoClient
import pprint

client = MongoClient('localhost', 27017)

db = client.test

mongodb_project = db.final_project.find_one()
```

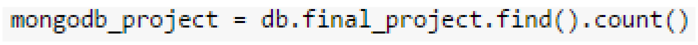
1. Statistical Overview of the Map Data

Size of the Files

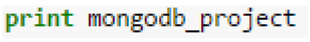
File Name	File Size (MBs)
minneapolis.osm	52.4
minneapolis.osm.json	56.3

Number of documents

Pymongo command:



```python
mongodb_project = db.final_project.find().count()
```



```python
print mongodb_project
```

Output:

```
260981
```

Number of nodes

Pymongo command:

```
mongodb_project = db.final_project.find({"type" : "node"}).count()

print mongodb_project
```

Output:

```
224607
```

Number of ways

Pymongo command:

```
mongodb_project = db.final_project.find({"type" : "way"}).count()

print mongodb_project
```

Output:

```
36374
```

Number of unique users

NOTE: 2 different methods were used to get the same result:

Pymongo command:

```
mongodb_project = db.final_project.distinct("created.user")

count = 0
for m in mongodb_project:
    count += 1

print count
```

MongoDB Shell Command:

```
> db.final_project.distinct("created.user").length
```

Output:

```
316
```

2. Some Additional Queries

Top 5 contributing users

Pymongo command:

```
mongodb_project = db.final_project.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                                    {"$sort":{"count":-1}}, {"$limit":5}])
```

```
pprint.pprint(mongodb_project)
```

Output:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'Mulad', u'count': 101201},
             {u'_id': u'iandees', u'count': 72293},
             {u'_id': u'sota767', u'count': 18674},
             {u'_id': u'DavidF', u'count': 10638},
             {u'_id': u'neuhausr', u'count': 9632}]}
```

Top 10 amenities in Minneapolis

Pymongo command:

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

```
pprint.pprint(mongodb_project)
```

Output:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'parking', u'count': 1640},
             {u'_id': u'restaurant', u'count': 175},
             {u'_id': u'bicycle_rental', u'count': 101},
             {u'_id': u'bench', u'count': 82},
             {u'_id': u'place_of_worship', u'count': 78},
             {u'_id': u'fast_food', u'count': 75},
             {u'_id': u'cafe', u'count': 74},
             {u'_id': u'school', u'count': 53},
             {u'_id': u'pub', u'count': 47},
             {u'_id': u'fuel', u'count': 41}]}
```

Number of users with only 1 post

Pymongo command:

```
mongodb_project = db.final_project.aggregate([{"$group":{"_id":"$created.user", "count":{"$sum":1}}},
                                              {"$group":{"_id":"$count", "num_users":{"$sum":1}}},
                                              {"$sort":{"_id":1}}, {"$limit":1}])
```

```
pprint.pprint(mongodb_project)
```

Output:

```
{u'ok': 1.0, u'result': [{u'_id': 1, u'num_users': 71}]}
```

Bank with most branches

Pymongo command:

```
mongodb_project = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"operator":{"$exists":1}, "amenity":"bank"}},
                                              {"$group":{"_id":"$operator", "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}}, {"$limit":1}])
```

```
pprint.pprint(mongodb_project)
```

Output:

```
{u'ok': 1.0, u'result': [{u'_id': u'Wells Fargo', u'count': 5}]}
```

NOTE: No surprise, Wells Fargo being the biggest bank in the state with no other major bank present such as Bank of America, Chase, Citibank, etc.

Most Gas Stations

Pymongo command:

```
mongodb_project = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"name":{"$exists":1},"amenity":"fuel"}},
                                              {"$group":{"_id":"$name", "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}}, {"$limit":3}])
```

```
pprint.pprint(mongodb_project)
```

Output:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'Holiday', u'count': 3},
             {u'_id': u'Super America', u'count': 2},
             {u'_id': u'Shell', u'count': 1}]}
```

5. Other Ideas about the Data Set

1. Additional Improvements in viewing the Map Dataset as json

Some additional items were addressed pertaining specifically to minneapolis.osm. This was done to improve the way information was represented for map data in json format. Extra code was written as part of the modified_data.py program which reflected the improvements in the final json document

Improvements made in the data are as follows:

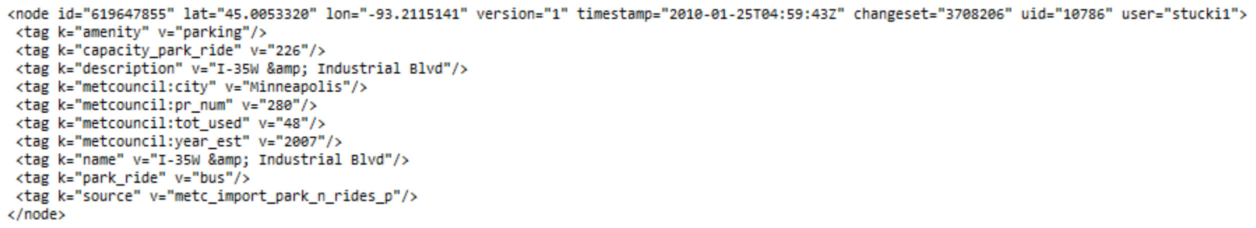
1. ‘Metcouncil’ [Metropolitan Council] sub dictionary was created in order to group the pertinent information.
2. A sub dictionary, ‘nice_ride_mn’ was created to represent useful information for bike rental services rendered by Nice Ride Minnesota [NRM].
3. ‘Bicycle_service’ sub dictionary was created to consolidate the information for a particular bicycle service/shop.
4. A Tiger database sub dictionary was created to depict the Topologically Integrated Geographic Encoding and Referencing [TIGER] data.
5. ‘Metro_parks’ sub dictionary was created to reflect the info for parks in the Minneapolis area.
6. ‘GNIS’ sub dictionary created to store information for the ‘ways’ that include water resources.
7. ‘location’ sub dictionary was created to include useful location information under the ‘is_in’ tag.
8. ‘UofMN” sub dictionary was created to include useful campus information for University of Minnesota.
9. Changes in ‘process map ()’ method in data.py to include non-latin names in the json format.
10. A sub dictionary for contact was made to include information such as phone, email, website, fax, etc.
11. ‘tree_type’ dictionary created to include types of trees planted in Minneapolis.

Details of the improvement changes implemented are as follows:

1. **Metcouncil [Metropolitan Council] dictionary was created in order to group the pertinent information.**

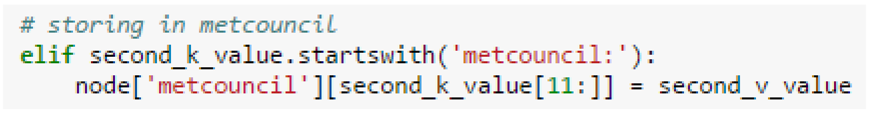
   http://www.metrocouncil.org/About-Us/The-Council-Who-We-Are.aspx

   Input in osm:



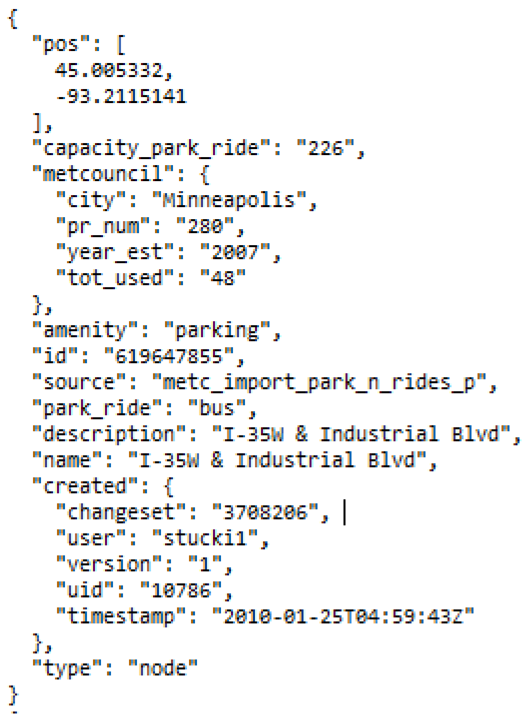
```
<node id="619647855" lat="45.0053320" lon="-93.2115141" version="1" timestamp="2010-01-25T04:59:43Z" changeset="3708206" uid="10786" user="stucki1">
 <tag k="amenity" v="parking"/>
 <tag k="capacity_park_ride" v="226"/>
 <tag k="description" v="I-35W &amp; Industrial Blvd"/>
 <tag k="metcouncil:city" v="Minneapolis"/>
 <tag k="metcouncil:pr_num" v="280"/>
 <tag k="metcouncil:tot_used" v="48"/>
 <tag k="metcouncil:year_est" v="2007"/>
 <tag k="name" v="I-35W &amp; Industrial Blvd"/>
 <tag k="park_ride" v="bus"/>
 <tag k="source" v="metc_import_park_n_rides_p"/>
</node>
```

   Code Excerpt from modified_data.py:



```python
# storing in metcouncil
elif second_k_value.startswith('metcouncil:'):
    node['metcouncil'][second_k_value[11:]] = second_v_value
```

   Output in json:



```
{
  "pos": [
    45.005332,
    -93.2115141
  ],
  "capacity_park_ride": "226",
  "metcouncil": {
    "city": "Minneapolis",
    "pr_num": "280",
    "year_est": "2007",
    "tot_used": "48"
  },
  "amenity": "parking",
  "id": "619647855",
  "source": "metc_import_park_n_rides_p",
  "park_ride": "bus",
  "description": "I-35W & Industrial Blvd",
  "name": "I-35W & Industrial Blvd",
  "created": {
    "changeset": "3708206", |
    "user": "stucki1",
    "version": "1",
    "uid": "10786",
    "timestamp": "2010-01-25T04:59:43Z"
  },
  "type": "node"
}
```

   NOTE: As per my understanding, above data could be useful to pull out information for different metropolitan council projects being done in Minneapolis.

2. **'nice_ride_mn' was created to represent useful information for bike rental services rendered by Nice Ride Minnesota [NRM].**

   https://www.niceridemn.org/

   Input in osm:

```
<node id="773403853" lat="44.9774060" lon="-93.2349110" version="3" timestamp="2011-10-02T03:03:29Z" changeset="9447377" uid="4732" user="iandees">
 <tag k="amenity" v="bicycle_rental"/>
 <tag k="capacity" v="12"/>
 <tag k="name" v="Williamson Hall"/>
 <tag k="nrm:id" v="27"/>
 <tag k="nrm:installed" v="true"/>
 <tag k="nrm:locked" v="false"/>
 <tag k="nrm:temporary" v="false"/>
 <tag k="nrm:terminal_name" v="30025"/>
</node>
```

   Code Excerpt from modified_data.py:

```
# storing in nice ride minnesota
elif second_k_value.startswith('nrm:'):
    node['nice_ride_mn'][second_k_value[4:]] = second_v_value
```

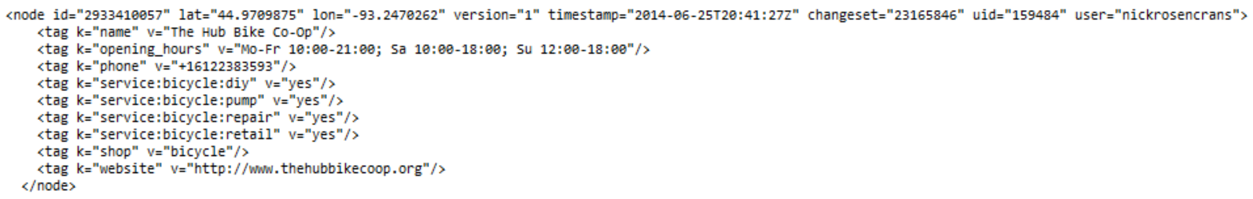
   Output in json:

```
{
  "nice_ride_mn": {
    "temporary": "false",
    "terminal_name": "30025",
    "locked": "false",
    "id": "27",
    "installed": "true"
  },
  "capacity": "12",
  "created": {
    "changeset": "9447377",
    "user": "iandees",
    "version": "3",
    "uid": "4732",
    "timestamp": "2011-10-02T03:03:29Z"
  },
  "name": "Williamson Hall",
  "pos": [
    44.977406,
    -93.234911
  ],
  "amenity": "bicycle_rental", |
  "type": "node",
  "id": "773403853"
}
```

   NOTE: The sub dictionary created above can be used to extract information regarding specific NRM rental spots. This sort of information can be useful for creating apps pertaining to cyclists.

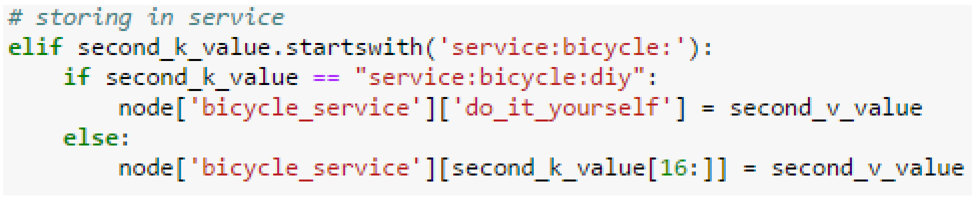
3. **‘Bicycle_service’ sub dictionary was created to consolidate the information for a particular bicycle service/shop. A ‘diy’ field was also modified as ‘do_it_yourself’ for easy readability.**

Input in osm:



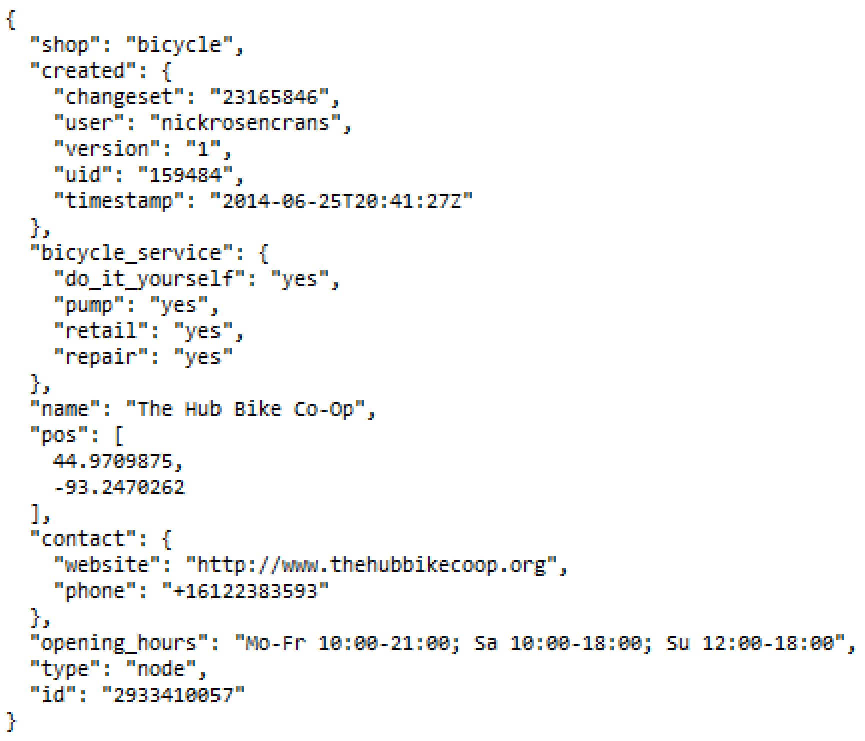
```
<node id="2933410057" lat="44.9709875" lon="-93.2470262" version="1" timestamp="2014-06-25T20:41:27Z" changeset="23165846" uid="159484" user="nickrosencrans">
    <tag k="name" v="The Hub Bike Co-Op"/>
    <tag k="opening_hours" v="Mo-Fr 10:00-21:00; Sa 10:00-18:00; Su 12:00-18:00"/>
    <tag k="phone" v="+16122383593"/>
    <tag k="service:bicycle:diy" v="yes"/>
    <tag k="service:bicycle:pump" v="yes"/>
    <tag k="service:bicycle:repair" v="yes"/>
    <tag k="service:bicycle:retail" v="yes"/>
    <tag k="shop" v="bicycle"/>
    <tag k="website" v="http://www.thehubbikecoop.org"/>
  </node>
```

Code Excerpt from modified_data.py:



```
# storing in service
elif second_k_value.startswith('service:bicycle:'):
    if second_k_value == "service:bicycle:diy":
        node['bicycle_service']['do_it_yourself'] = second_v_value
    else:
        node['bicycle_service'][second_k_value[16:]] = second_v_value
```

Output in json:

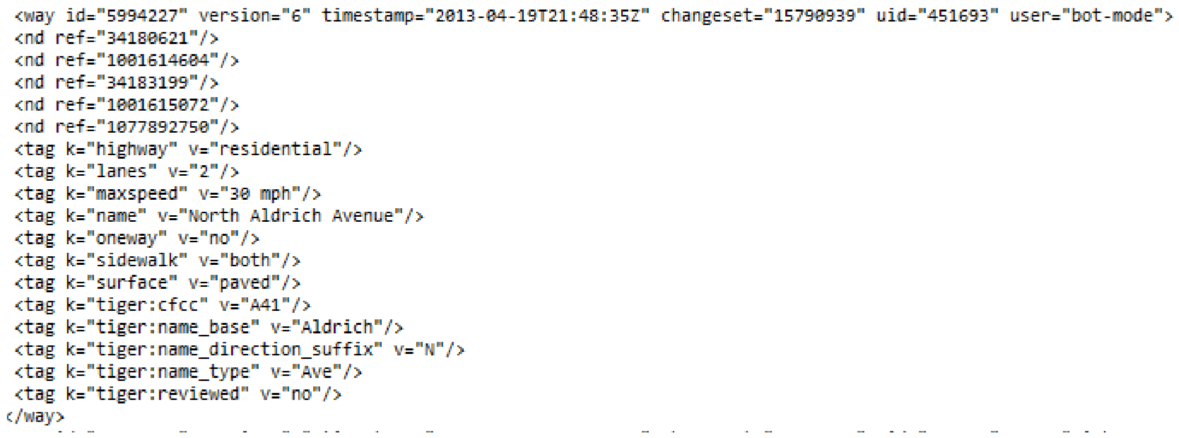


```
{
  "shop": "bicycle",
  "created": {
    "changeset": "23165846",
    "user": "nickrosencrans",
    "version": "1",
    "uid": "159484",
    "timestamp": "2014-06-25T20:41:27Z"
  },
  "bicycle_service": {
    "do_it_yourself": "yes",
    "pump": "yes",
    "retail": "yes",
    "repair": "yes"
  },
  "name": "The Hub Bike Co-Op",
  "pos": [
    44.9709875,
    -93.2470262
  ],
  "contact": {
    "website": "http://www.thehubbikecoop.org",
    "phone": "+16122383593"
  },
  "opening_hours": "Mo-Fr 10:00-21:00; Sa 10:00-18:00; Su 12:00-18:00",
  "type": "node",
  "id": "2933410057"
}
```

NOTE: This sub dictionary can be used to provide information to private bicycle owners about the specific service offered at a particular location. This information, if up to date in open street maps, can be very useful as an input for a cell phone app.

4. **Tiger database sub dictionary was created to depict the Topologically Integrated Geographic Encoding and Referencing [TIGER] data.**
   https://www.census.gov/geo/maps-data/data/tiger.html
   http://www.maris.state.ms.us/pdf/cfcccodes.pdf

Input in osm:

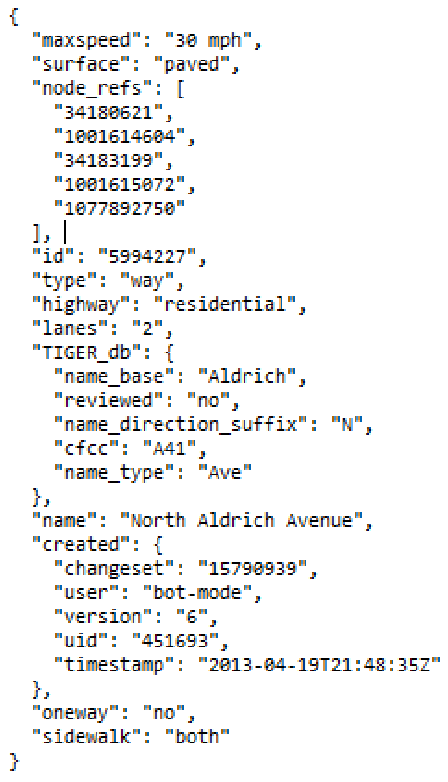


```
 <way id="5994227" version="6" timestamp="2013-04-19T21:48:35Z" changeset="15790939" uid="451693" user="bot-mode">
 <nd ref="34180621"/>
 <nd ref="1001614604"/>
 <nd ref="34183199"/>
 <nd ref="1001615072"/>
 <nd ref="1077892750"/>
 <tag k="highway" v="residential"/>
 <tag k="lanes" v="2"/>
 <tag k="maxspeed" v="30 mph"/>
 <tag k="name" v="North Aldrich Avenue"/>
 <tag k="oneway" v="no"/>
 <tag k="sidewalk" v="both"/>
 <tag k="surface" v="paved"/>
 <tag k="tiger:cfcc" v="A41"/>
 <tag k="tiger:name_base" v="Aldrich"/>
 <tag k="tiger:name_direction_suffix" v="N"/>
 <tag k="tiger:name_type" v="Ave"/>
 <tag k="tiger:reviewed" v="no"/>
</way>
```

Code Excerpt from modified_data.py:

```python
# storing in TIGER DB
elif second_k_value.startswith('tiger:'):
    node['TIGER_db'][second_k_value[6:]] = second_v_value
```

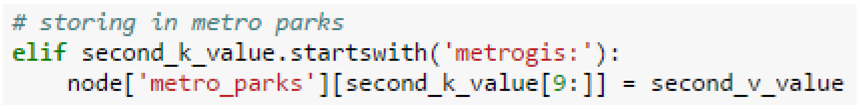
Output in json:



```
{
  "maxspeed": "30 mph",
  "surface": "paved",
  "node_refs": [
    "34180621",
    "1001614604",
    "34183199",
    "1001615072",
    "1077892750"
  ],
  "id": "5994227",
  "type": "way",
  "highway": "residential",
  "lanes": "2",
  "TIGER_db": {
    "name_base": "Aldrich",
    "reviewed": "no",
    "name_direction_suffix": "N",
    "cfcc": "A41",
    "name_type": "Ave"
  },
  "name": "North Aldrich Avenue",
  "created": {
    "changeset": "15790939",
    "user": "bot-mode",
    "version": "6",
    "uid": "451693",
    "timestamp": "2013-04-19T21:48:35Z"
  },
  "oneway": "no",
  "sidewalk": "both"
}
```

Rest of the improvements for displaying the data were done in the same fashion with conditions set as below:

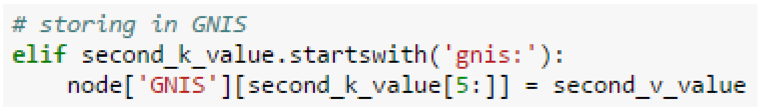
5. **‘Metro_parks’ sub dictionary was created to reflect the info for parks in the Minneapolis area.**



```
# storing in metro parks
elif second_k_value.startswith('metrogis:'):
    node['metro_parks'][second_k_value[9:]] = second_v_value
```

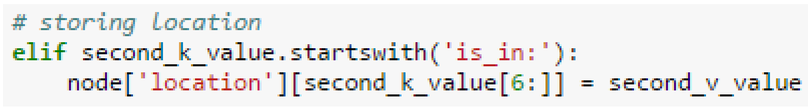
6. **‘GNIS’ sub dictionary was created to store information for the ‘ways’ that include water resources.**

   http://nhd.usgs.gov/gnis.html



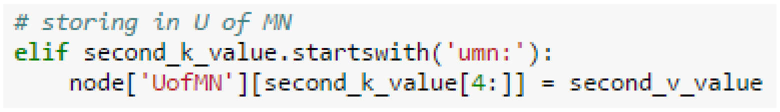
```
# storing in GNIS
elif second_k_value.startswith('gnis:'):
    node['GNIS'][second_k_value[5:]] = second_v_value
```

7. **‘location’ sub dictionary was created to include useful location information under the ‘is_in’ tag.**



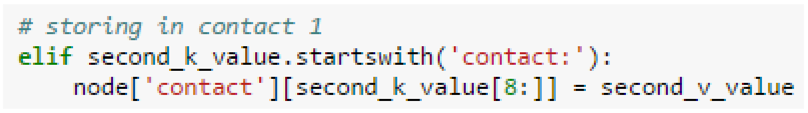
```
# storing location
elif second_k_value.startswith('is_in:'):
    node['location'][second_k_value[6:]] = second_v_value
```

8. **‘UofMN” sub dictionary was created to include useful campus information for University of Minnesota.**



```
# storing in U of MN
elif second_k_value.startswith('umn:'):
    node['UofMN'][second_k_value[4:]] = second_v_value
```

9. **A sub dictionary for contact was made to include information such as phone, email, website, fax, etc.**



```
# storing in contact 1
elif second_k_value.startswith('contact:'):
    node['contact'][second_k_value[8:]] = second_v_value
```

10. ‘tree_type’ dictionary created to include types of trees planted in Minneapolis.

This dictionary element was implemented to avoid duplication for regular tag type that was used to store node types such as ‘node’ and ‘way’.

Storing regular tag type for ‘node’ and ‘way’:

```python
def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :
        # YOUR CODE HERE

        node['type'] = element.tag # assigning node or way to type
```

Modification in code to accommodate tree types:

Input in osm:

```xml
<node id="1540351861" lat="44.9731889" lon="-93.2362028" version="2" timestamp="2013-04-05T20:04:07Z" changeset="15624633" uid="159484" user="nickrosencrans">
 <tag k="denotation" v="urban"/>
 <tag k="natural" v="tree"/>
 <tag k="type" v="broad_leaved"/>
</node>
```

Code Excerpt from modified_data.py:

```python
# storing in tree type
elif second_k_value == "type":
    node['tree_type'] = second_v_value
```

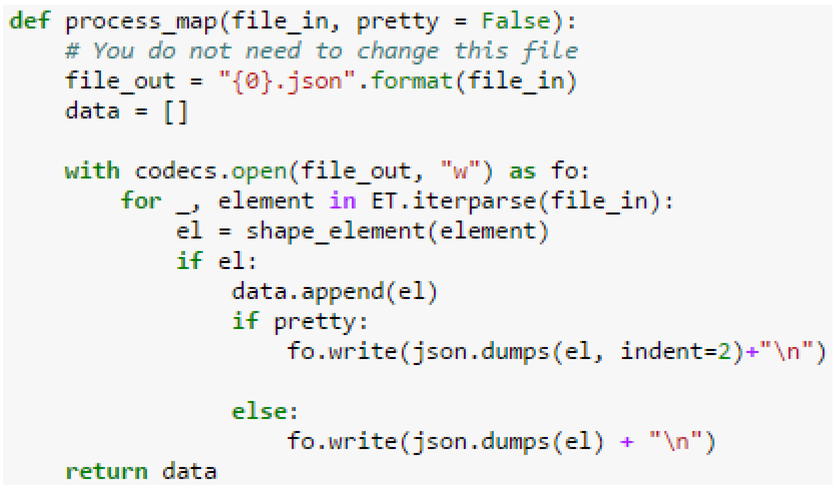
Output in json:

```json
{
  "natural": "tree",
  "created": {
    "changeset": "15624633",
    "user": "nickrosencrans",
    "version": "2",
    "uid": "159484",
    "timestamp": "2013-04-05T20:04:07Z"
  },
  "pos": [
    44.9731889,
    -93.2362028
  ],
  "tree_type": "broad_leaved",
  "denotation": "urban",
  "type": "node",
  "id": "1540351861"
}
```

11. Changes in ‘process map ()’ method in data.py to include non-latin names in the json format.

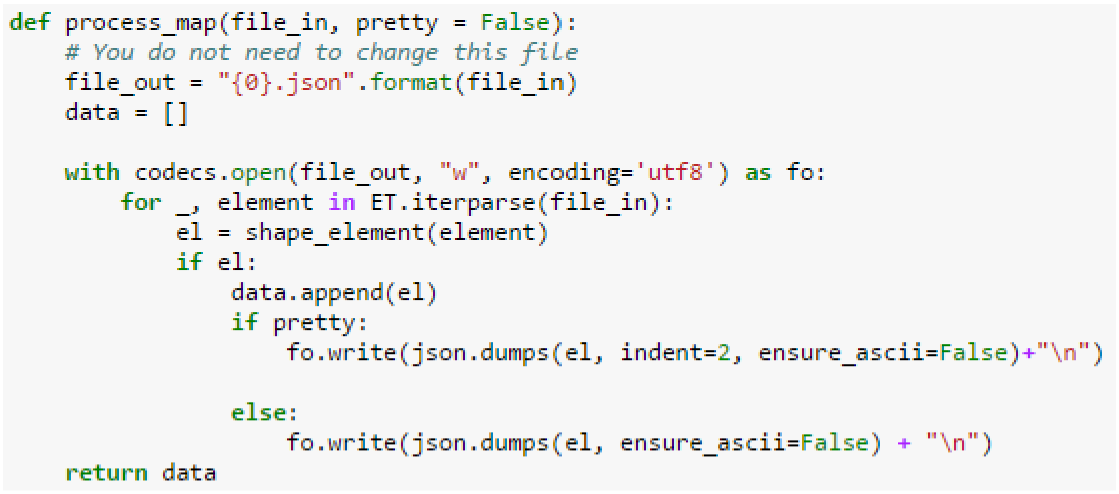
A modified version of process_map () method was implemented in modified_data.py to maintain the non-latin city name characters.

Original process map method implementation in data.py:



```python
def process_map(file_in, pretty = False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []

    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")

                else:
                    fo.write(json.dumps(el) + "\n")
    return data
```

Modified version of the process map method implemented in the modified_data.py program:



```python
def process_map(file_in, pretty = False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []

    with codecs.open(file_out, "w", encoding='utf8') as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2, ensure_ascii=False)+"\n")

                else:
                    fo.write(json.dumps(el, ensure_ascii=False) + "\n")
    return data
```

2. *Additional data exploration from the Map using MongoDB*

Top 5 places of worship

Pymongo command:

```
mongodb_project = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"religion":{"$exists":1},"amenity":"place_of_worship"}},
                                              {"$group":{"_id":"$religion", "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}}, {"$limit":5}])
```

```
pprint.pprint(mongodb_project)
```

Output:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'christian', u'count': 60},
             {u'_id': u'muslim', u'count': 3},
             {u'_id': u'buddhist', u'count': 3},
             {u'_id': u'unitarian_universalist', u'count': 1},
             {u'_id': u'jewish', u'count': 1}]}
```

NOTE: The 2nd and 3rd most worshiped religion is consistent with what meets the eye. There is abundance of Somalis (Muslim) and Tibetans (Buddhist) refugees who have settled in Minneapolis over the years.

Most tree plantations

Pymongo command:

```
mongodb_project = db.final_project.aggregate([{"$match":{"natural":{"$exists":1},"tree_type":{"$exists":1},"natural":"tree"}},
                                              {"$group":{"_id":"$tree_type", "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}}, {"$limit":3}])
```

```
pprint.pprint(mongodb_project)
```

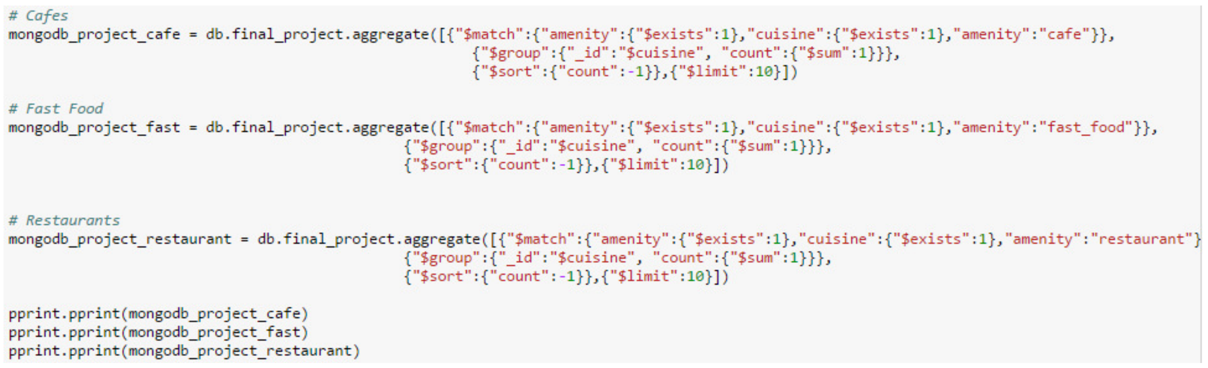
Output:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'broad_leaved', u'count': 2115},
             {u'_id': u'conifer', u'count': 450},
             {u'_id': u'coniferous', u'count': 123}]}
```

NOTE: The tree_type sub dictionary was used to build the above query and generate the results via pymongo.

Top 10 popular cuisines (cafes, fast food & restaurants)

Pymongo commands:



```
# Cafes
mongodb_project_cafe = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"cuisine":{"$exists":1},"amenity":"cafe"}},
                                                    {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
                                                    {"$sort":{"count":-1}},{"$limit":10}])

# Fast Food
mongodb_project_fast = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"cuisine":{"$exists":1},"amenity":"fast_food"}},
                                            {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
                                            {"$sort":{"count":-1}},{"$limit":10}])


# Restaurants
mongodb_project_restaurant = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"cuisine":{"$exists":1},"amenity":"restaurant"}
                                            {"$group":{"_id":"$cuisine", "count":{"$sum":1}}},
                                            {"$sort":{"count":-1}},{"$limit":10}])

pprint.pprint(mongodb_project_cafe)
pprint.pprint(mongodb_project_fast)
pprint.pprint(mongodb_project_restaurant)
```

Output:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'coffee_shop', u'count': 23},
             {u'_id': u'coffee', u'count': 4},
             {u'_id': u'chinese', u'count': 1},
             {u'_id': u'ice_cream', u'count': 1},
             {u'_id': u'donut', u'count': 1},
             {u'_id': u'Tea', u'count': 1},
             {u'_id': u'meat', u'count': 1}]}
{u'ok': 1.0,
 u'result': [{u'_id': u'sandwich', u'count': 17},
             {u'_id': u'burger', u'count': 12},
             {u'_id': u'pizza', u'count': 6},
             {u'_id': u'mexican', u'count': 6},
             {u'_id': u'kebab', u'count': 2},
             {u'_id': u'ice_cream', u'count': 2},
             {u'_id': u'americanized_mexican', u'count': 1},
             {u'_id': u'mediterranean;ice_cream', u'count': 1},
             {u'_id': u'american', u'count': 1},
             {u'_id': u'chinese', u'count': 1}]}
{u'ok': 1.0,
 u'result': [{u'_id': u'pizza', u'count': 14},
             {u'_id': u'chinese', u'count': 11},
             {u'_id': u'american', u'count': 7},
             {u'_id': u'italian', u'count': 6},
             {u'_id': u'vietnamese', u'count': 6},
             {u'_id': u'thai', u'count': 5},
             {u'_id': u'regional', u'count': 5},
             {u'_id': u'sandwich', u'count': 4},
             {u'_id': u'sushi', u'count': 4},
             {u'_id': u'french', u'count': 3}]}
```

NOTE: Some of the above are interchangeable cuisines such as Pizza, American, Ice cream and Chinese.

Top 5 Bike rentals spots with most capacity

Pymongo command:

```
mongodb_project = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"capacity":{"$exists":1},"amenity":"bicycle_rental"}},
                                              {"$project":{"_id":"$name", "capacity":"$capacity"}},
                                              {"$sort":{"capacity":-1}},{"$limit":5}])
```

```
pprint.pprint(mongodb_project)
```

Output:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'Coffman Memorial Union Station', u'capacity': u'35'},
             {u'_id': u'Social Sciences', u'capacity': u'33'},
             {u'_id': u'19th Ave.', u'capacity': u'31'},
             {u'_id': u'3rd Ave. S', u'capacity': u'27'},
             {u'_id': u'2nd Ave. S', u'capacity': u'27'}]}
```

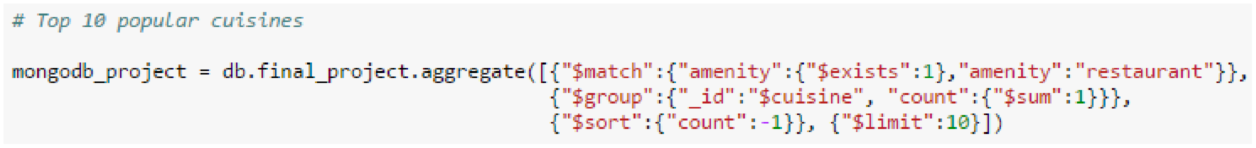
3. *Additional observation about the Map Data*

‘None’ result generated due to no existing sub field in dictionary

MongoDB queries were modified to check for non-existent sub field and generate results excluding ‘None’.

Example #1 (Top 10 popular cuisines):

Pymongo command without modification:

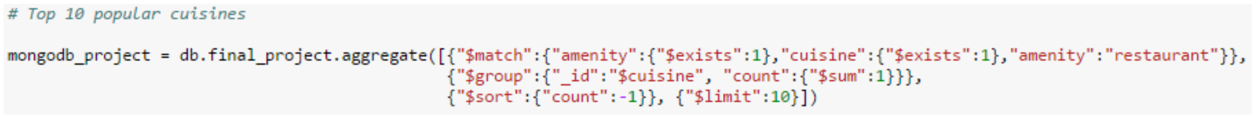


```
# Top 10 popular cuisines

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

Output without modification:

```
{u'ok': 1.0,
 u'result': [{u'_id': None, u'count': 69},
             {u'_id': u'pizza', u'count': 14},
             {u'_id': u'chinese', u'count': 11},
             {u'_id': u'american', u'count': 7},
             {u'_id': u'italian', u'count': 6},
             {u'_id': u'vietnamese', u'count': 6},
             {u'_id': u'thai', u'count': 5},
             {u'_id': u'regional', u'count': 5},
             {u'_id': u'sushi', u'count': 4},
             {u'_id': u'sandwich', u'count': 4}]}
```

Pymongo command with modification:



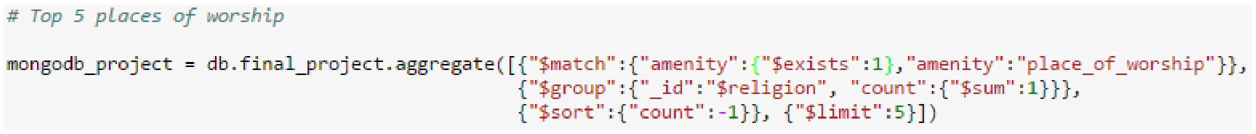
```
# Top 10 popular cuisines

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

Output with modification:

```
{u'ok': 1.0,
 u'result': [{u'_id': u'pizza', u'count': 14},
             {u'_id': u'chinese', u'count': 11},
             {u'_id': u'american', u'count': 7},
             {u'_id': u'italian', u'count': 6},
             {u'_id': u'vietnamese', u'count': 6},
             {u'_id': u'thai', u'count': 5},
             {u'_id': u'regional', u'count': 5},
             {u'_id': u'sandwich', u'count': 4},
             {u'_id': u'sushi', u'count': 4},
             {u'_id': u'french', u'count': 3}]}
```

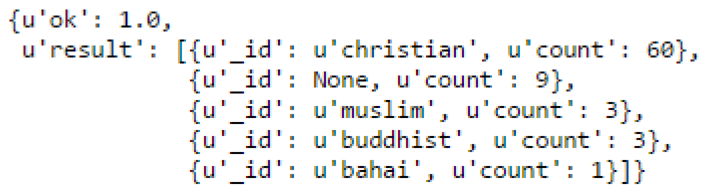
Example #2 (Top 5 places of worship):

Pymongo command without modification:



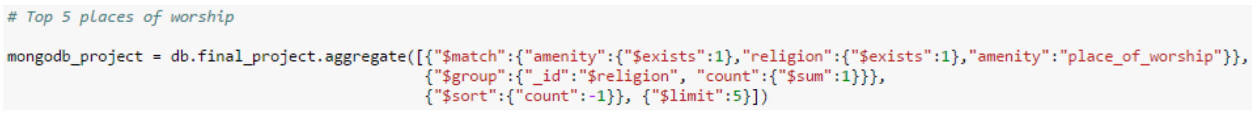
```
# Top 5 places of worship

mongodb_project = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"amenity":"place_of_worship"}},
                                              {"$group":{"_id":"$religion", "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}}, {"$limit":5}])
```

Output without modification:



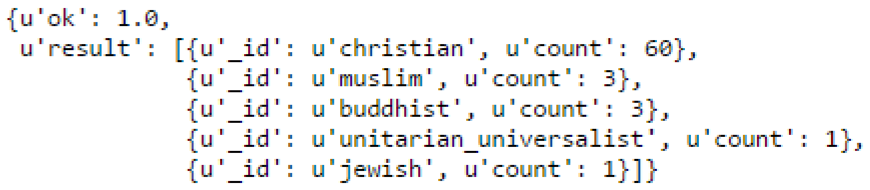
```
{u'ok': 1.0,
 u'result': [{u'_id': u'christian', u'count': 60},
             {u'_id': None, u'count': 9},
             {u'_id': u'muslim', u'count': 3},
             {u'_id': u'buddhist', u'count': 3},
             {u'_id': u'bahai', u'count': 1}]}
```

Pymongo command with modification:



```
# Top 5 places of worship

mongodb_project = db.final_project.aggregate([{"$match":{"amenity":{"$exists":1},"religion":{"$exists":1},"amenity":"place_of_worship"}},
                                              {"$group":{"_id":"$religion", "count":{"$sum":1}}},
                                              {"$sort":{"count":-1}}, {"$limit":5}])
```

Output with modification:



```
{u'ok': 1.0,
 u'result': [{u'_id': u'christian', u'count': 60},
             {u'_id': u'muslim', u'count': 3},
             {u'_id': u'buddhist', u'count': 3},
             {u'_id': u'unitarian_universalist', u'count': 1},
             {u'_id': u'jewish', u'count': 1}]}
```

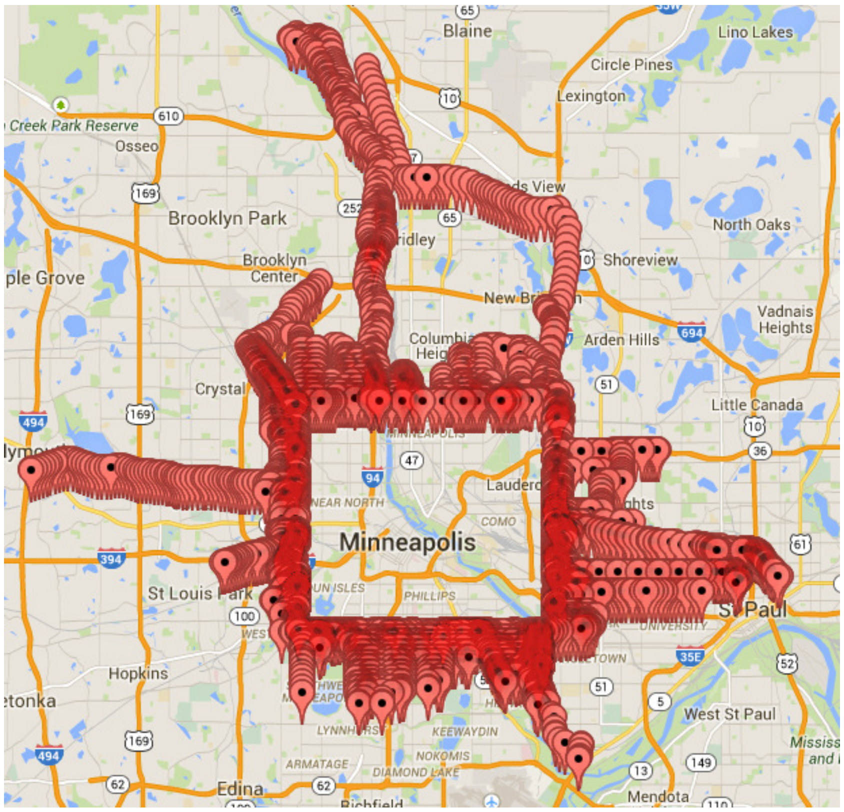
Incorrect coordinates for ‘nodes’

Following piece of code was run to check if nodes created with in the map boundary had correct coordinates. Then the resulting coordinates were plotted in google map. MS Excel sheet (Nodes outside map boundary.xlsx is in the zip file for reference).

Python code:

```python
def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :
        # YOUR CODE HERE

        node['type'] = element.tag # assigning node or way to type

        node['created'] = {} # intializing created

        for k_value, v_value in element.items(): # iterating through the element

            if k_value == "lat" or k_value == "lon": # checking for lat and lon

                lat = element.get('lat')
                lon = element.get('lon')
                user = element.get('user')
                if lon is not None and lat is not None: # converting to float
                    float_lat = float(lat)
                    float_lon = float(lon)

                    if (float_lat > 45.0168 or float_lon < -93.3206 or float_lat < 44.9347 or float_lon > -93.1908):
                        print float_lat,float_lon, user,node['type']
```

Output (some excel manipulation and plotting in google maps):



6. Conclusion

A detailed review was conducted for the selected area of Minneapolis, MN and it was observed to have certain shortcomings. I tried to audit, clean, correct, standardize the data in a manner to make it presentable (by adding certain sub dictionaries, code modifications, name updates, etc) to the best of my ability.

Additional observations about the inconsistencies in the data were noted and possible work around/resolutions were provided by generating a more enhanced version of the shape_element () method of the data.py program (also including modified version of the update_name () method from audit.py program). Further auditing and cleaning can be done to make the map street data more consistent and uniform with the help of the most contributing users in Minneapolis area.