Section 1: Exploring the data

Using the mapparser.py file I was able to determine the tags which were included within the Open Street Map file as well as how many times they appeared. Below is the output from that file listed in a python dictionary.

{'bounds': 1,
 'member': 5033,
 'nd': 716616,
 'node': 591720,
 'osm': 1,
 'relation': 408,
 'tag': 401475,
 'way': 63741}

As you can see there are over 400,000 tags, which will give an indication of what of entity exists at a particular location. This will be helpful in running queries of a few specific entities when we examine our dataset and further examine the nodes and ways. Using the OSM Wiki we know that the 'ways' of the dataset are the 'streets' from the map. According to the data above we know there are over 63,000 'streets' within the OSM dataset.

Next I used the tags.py file to examine the "k" value for each "<tag>". This will help me identify potential problem tags as well as determine if I can use these tags as keys in MongoDB. The output of this file is listed below.

{'lower': 161471, 'lower_colon': 229169, 'other': 10835, 'problemchars': 0}

From this we know there are no problem characters which were identified.

The users.py file explores the number of unique users which have contributed to the OSM datafile. The output of the file indicates there are 587 unique users (identified by the uid tag). These users are also printed with the output.

After exploring the data and gaining a basic understanding of what I'm working with I ran audit.py to print out a list of street names which were not included within my "expected" list. In other words I'm using this file to identify all the street names which I might want to fix prior to prepping my OSM datafile to be imported into MongoDB. Below is my initial output of "unexpected" streets from the audit.py output.

{'164': set(['HWY 164', 'N6620 HWY 164', 'State Road 164']),
 '41': set(['Highway 41']),
 '53076': set(['53076']),
 '83': set(['North Highway 83']),
 'Ave': set(['E Wisconsin Ave',
 'Milwaukee Ave',
 'N. Prospect Ave',
 'North Murray Ave',
 'North Summit Ave',
 'S Howell Ave',

'Summit Ave',
'W Appleton Ave',
'W Fond du Lac Ave',
'W Grand Ave',
'W Layton Ave',
'Wisconsin Ave']),
'Ave.': set(['E. Garfield Ave.',
'E. North Ave.',
'E. Thomas Ave.',
'N. Humboldt Ave.',
'N. Prospect Ave.',
'N. Summit Ave.',
'North Oakland Ave.',
'W. Michigan Ave.',
'W. Wisconsin Ave.']),
'Blvd': set(['East Kensington Blvd', 'N Grandview Blvd', 'W Highland Blvd']),
'Broadway': set(['N. Broadway']),
'Ct': set(['Manchester Ct', 'N16880 Tillie lake Ct', 'W Oakwood Part Ct']),
'Dr': set(['E Capitol Dr',
'N9581 Bancroft Dr',
'W Schroeder Dr',
'W Sunset Dr']),
'Dr.': set(['N. Lincoln Memorial Dr.']),
'Humboldt': set(['N. Humboldt']),
'NN': set(['County Highway NN']),
'O': set(['County Road O']),
'Pkwy': set(['Miller Pkwy']),
'Pl.': set(['E. Kenilworth Pl.']),
'Rd': set(['Golf Rd', 'N Port Washington Rd', 'West County Line Rd']),
'Rd.': set(['E. Reservoir Rd.',
'N. Industrial Rd.',
'N. Port Washington Rd.',
'N. Riverboat Rd.']),
'St': set(['E Sumner St',
'N 124th St',
'N 9th St',
'N Pine St',
'N Rochester St',
'N Weil St',
'S 13th St',
'West Galena St']),
'St.': set(['E. Center St.',
'E. Locust St.',
'N. Bremen St.',
'N. Commerce St.',
'N. Hubbard St.',
'N. Humboldt St.',
'N. Water St.',

```
        'N. Weil St.',
        'S. Jefferson St.',
        'W. Pleasant St.']),
 'W': set(['Highway W']),
 'Way': set(['North Riverwalk Way', 'West Freshwater Way']),
 'West': set(['Highway 167 West']),
 'Wright': set(['E. Wright'])}
```

From this output I can develop a "mapping list" to utilize in my next step for updating my street names. I have highlighted the abbreviated street names I will include within this mapping. The ones not highlighted I am not concerned with as these are part of actual street names. The numbers and letters within this list are typical of highway names in Milwaukee such as Highway 83 or County Road N. I also forgot about adding 'Way' to the expected name list so I will update this on my next pass. (After adding 'Way' to the expected list and re-running audit.py, this 2 streets were removed from the list of "unexpected" street names.

The next step in my analysis is to clean up the street name abbreviations which were identified in the previous step. Adding them to my "mapping" list in the cleandata.py file, the output is shown below, which shows the results of my data scrub. I've included an assertion within the cleandata.py file to print out the results of my data scrub and ensure Milwaukee Ave was corrected to Milwaukee Avenue as well as Miller Pkwy was replaced with Miller Parkway.

In reviewing the list is also appears a website tag "napaonline.com" was inserted into a "way" field. I know for a fact there are no street names in Milwaukee called napaonline.com. Since this is an isolated error I will leave the record alone. Below is an excerpt of the output from the cleandata.py file. The highlighted "Pass" at the end of the results indicates the 2 changes I anticipated and set assertions for were met.

W Layton Ave => W Layton Avenue
West Forest Home Ave => West Forest Home Avenue
S7959 Racine Ave => S7959 Racine Avenue
Harwood Ave => Harwood Avenue
N. Oakland Ave => N. Oakland Avenue
Milwaukee Ave => Milwaukee Avenue
North Summit Ave => North Summit Avenue
N. Prospect Ave => N. Prospect Avenue
W National Ave => W National Avenue
E Juneau Ave => E Juneau Avenue
W North Ave => W North Avenue
W Appleton Ave => W Appleton Avenue
W Fond du Lac Ave => W Fond du Lac Avenue
Summit Ave => Summit Avenue
E Wisconsin Ave => E Wisconsin Avenue
Washington Ave => Washington Avenue
W Forest Home Ave => W Forest Home Avenue
Wisconsin Ave => Wisconsin Avenue
W Grand Ave => W Grand Avenue
North Murray Ave => North Murray Avenue

S Howell Ave => S Howell Avenue
South Kinnickinnic Ave => South Kinnickinnic Avenue
HWY 164 => HWY 164
State Road 164 => State Road 164
N6620 HWY 164 => N6620 HWY 164
McCanna Pkwy => McCanna Parkway
Miller Pkwy => Miller Parkway
Kettle Moraine Drive South => Kettle Moraine Drive South
N. Humboldt => N. Humboldt
Highway 18 => Highway 18
West Brown Deer => West Brown Deer
East Kensington Blvd => East Kensington Boulevard
N Grandview Blvd => N Grandview Boulevard
W68N611 Evergreen Blvd => W68N611 Evergreen Boulevard
W Highland Blvd => W Highland Boulevard
E Moreland Blvd => E Moreland Boulevard
N Broadway => N Broadway
N Broadway => N Broadway
N. Broadway => N. Broadway
napaonline.com => napaonline.com
Scenic Ct => Scenic Court
W Oakwood Part Ct => W Oakwood Part Court
Wegge Ct => Wegge Court
Manchester Ct => Manchester Court
N16880 Tillie lake Ct => N16880 Tillie lake Court
Pass

The next task is to shape the data into a dictionary format to and prepare the data to be saved into JSON format. This sets up the next task for me to load the data into MongoDB and start to run queries. Using the solution from Problem Set 6 I was able to format the data according to the requirements set forth in the lesson plan. Below is one of the documents from the JSON file that was output from the data.py file.

{'created': {'changeset': '189529',
'timestamp': '2007-08-04T20:44:03Z',
'uid': '979',
'user': 'Hawke',
'version': '1'},
'id': '19775999',
'pos': [43.0123289, -87.8938094],
'type': 'node'}

The next step is to load my json file (milwaukee_wisconsin.osm.json) into MongoDB. The file named mongodb.py is responsible for loading the data into MongoDB – into a database named milwaukee. Once loaded into the MongoDB database I ran the code show dbs to display the databases. “milwaukee” came up as the database and was listed as .453GB. So I know the documents were successfully loaded in MongoDB so my next step is to run queries of the data.

Section 2: Querying the data
From the MongoDB shell I ran a few simple queries to get more info on my data.

Total number of entries:
db.milwaukee.find().count()
number of elements: 655,461

Total number of 'node' entries:
db.milwaukee.find({"type":"node"}).count()
number of node elements: 591,711

Total number of 'way' entries:
db.milwaukee.find({"type":"way"}).count()
number of way elements: 63,736

Since Milwaukee is known for its bars – I ran the next query to see how many amenities were listed as "bars".
db.milwaukee.find({"amenity":"bar"}).count()
number of bars: 63

I also wrote another program, queries.py to run more complex queries. These queries include the number of unique users, number of users with more than 1 entry, the top 10 users contributing to the data updates, the top 5 types of shops and the top 5 amenities. Below is the output from these queries along with the code for each query.

milwaukee.distinct("created.user").length
number of unique users - 582

db.milwaukee.aggregate([{"$group":{"_id":"$created.user","posts":{"$sum":1}}},{"$match":{"posts":{"$gt":1}}},{"$group":{"_id":"users","count":{"$sum":1}}}])
460 users have more than one entry

db.milwaukee.aggregate([{"$match":{"created.user":{"$exists":1}}},{"$group":{"_id":"$created.user","count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":10}])
woodpeck_fixbot made 184680 updates
Gary Cox made 35052 updates
ItalianMustache made 34627 updates
bbauter made 33876 updates
iandees made 28806 updates
hogrod made 26341 updates
TIGERcnl made 22142 updates
Rub21 made 21987 updates
Mulad made 20154 updates
Jertel made 19977 updates

milwaukee.aggregate([{"$match":{"shop":{"$exists":1}}},{"$group":{"_id":"$shop","count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":5}])
92 supermarket stores

46 clothes stores
46 convenience stores
29 car_repair stores
27 doityourself stores

db.milwaukee.aggregate([{"$match":{"amenity":{"$exists":1}}},
{"$group":{"_id":"$amenity","count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":5}])
1253 parking
857 school
290 restaurant
177 fast_food
157 grave_yard

The file size for my data files are as follows:
milwaukee_wisconsin.osm = 126MB
milwaukee_wisconsin.osm.json = 143MB

Section 3: Other ideas for the dataset

In running the query for "bars" and finding "only" 63 sparked my interest. I know there are well over 100 bars in Milwaukee and a future project could be done to update this data. A potential issue with this update would be found when some bars also consider themselves as restaurants. This type of metadata clarification would be necessary to ensure only bars were listed if we wanted only bars and not bar/restaurants. Once complete a user could run a geospatial query to find all bars within a particular area of the city. This could be useful in planning a night out on the town!

Section 4: References
Udacity user forums and class lessons
http://wiki.openstreetmap.org/wiki/OSM_XML