Nanodegree: Data Science

Project #2 – Data Wrangling with MondoDB

Tim Tse

4/1/2015

**Project Summary**

What is your name?

Tim Tse

What E-mail address do you use to sign in to Udacity?

[tt757x@att.com](mailto:tt757x@att.com)

What area of the world you used for your project? Post a link to the map position and write a short description. Note that the osm file of the map should be at least 50MB.

URL: <https://www.openstreetmap.org/relation/3529865#map=12/34.1048/-118.0656>

I picked an area within Los Angles for my project. I choose this particular place because this is where I grew up in. I am familiar with the surrounding areas and it would be fun to explore it in my project.

Is there a list of  Web sites, books, forums, blog posts, github repositories etc that you referred to or used  in this  submission (Add N/A if you did not use  such resources)?

<http://discussions.udacity.com/c/nd002-2014-11-05>

<http://blog.zagwozdka.com/2014/03/udacity-data-wrangling-with-mongo-db.html>

<https://github.com/FlyingDad/UD032-Data-Wrangling-with-MongoDB>

<https://piazza.com/class/i23uptiifb6194?cid=679>

<https://www.youtube.com/watch?v=NNCahQYZ9-s>

<http://www.jasondamiani.com/portfolio/data-wrangling-openstreetmaps-data/>

<https://github.com/paul-reiners/udacity-data-wrangling-mongo-db>

<https://www.youtube.com/watch?v=LDCKzIwpirM>

Please carefully read the following statement and include it in your email:

*“I hereby confirm that this submission is my work. I have cited above the origins of****any****parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc. By including this in my email, I understand that I will be expected to explain my work in a video call with a Udacity coach before I can receive my verified certificate.”*

Is there any other important information that you would want your project evaluator to know?

No

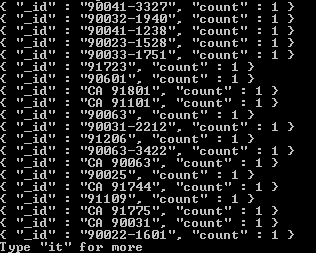
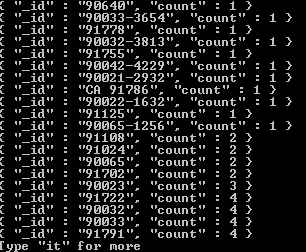
**Project Report**

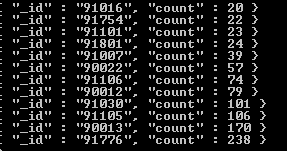
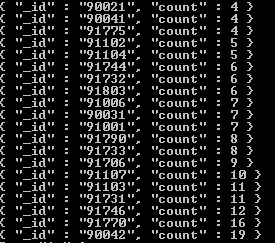
1. Problems encountered in my map

After downloading the map of the San Garbiel Valley area in California and running it against the data.py file created in lesson 6, I noticed three main problems with the dataset. I will discuss the three problems in this session.

1. The dataset consists of different format of postal codes. From the output, we can see there are three different formats of postal codes. They are “CA 91744”, “91744”, and “90033-1751”.

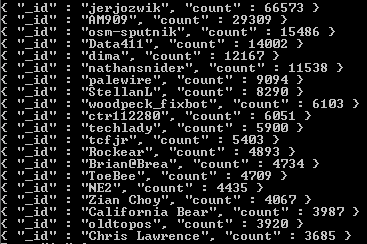
* db.map.aggregate([{"$match":{"address.postcode":{"$exists":1}}}, {"$group":{"\_id":"$address.postcode", "count":{"$sum":1}}}, {"$sort":{"count":­1}}])





1. The contributions of users are skewed. We can see the top user “jerjozwik” contributed more than twice than the second user. Although there are a total of 503 users that inputted into this dataset, the top 20 users contributed most of the map updates.

* db.map.aggregate([{"$group":{"\_id":"$created.user", "count":{"$sum":1}}}, {"$sort":{"count":­-1}}])



1. Missing data. The following code shows the top cuisine in the dataset. We can see that there are 117 data with missing data. There are 25 cuisine entries that are tagged as “Chinese”. However, we cannot conclude that Chinese cuisine is the most popular cuisine due to the high number of missing data.

* db.map.aggregate([{"$match":{"amenity":{"$exists":1},”amenity”:”restaurant”}}, {"$group":{"\_id":"$cuisine", "count":{"$sum":1}}}, {"$sort":{"count":­-1}}])



1. DataOverview

This section contains basic statistics for the temple city map dataset and the MongoDB queries used to obtain the information.

The data.py from lesson 6 prepared the dataset “example.osm” to “example.osm.json”

Mongoimport –db map –collection map –file example.osm.json

File size

* db.map.dataSize()
* 104659232 : 104.6 MB

# Number of nodes

* db.map.find({“type”:”node”}).count()
* 295587

# Number of ways

* db.map.find({“type”:”way”}).count()
* 43693

# Number of unique users

* db.map.distinct(“created.user”).length
* 503

#Number of address

* db.map.find({“address”:{$exists:true}}).count()
* 57953

# Number of unique version

* db.map.distinct(“created.version”).length
* 94

1. Additional Ideas

# Popular amenity

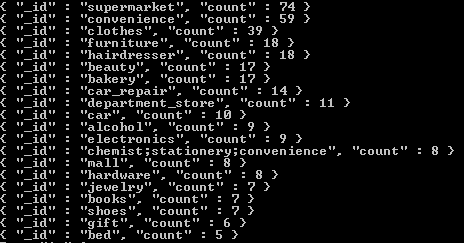
* db.map.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"\_id":"$amenity ", "count":{"$sum":1}}}, {"$sort":{"count":­-1}}])



* I am surprise to find that “place to worship” is the top 3 amenity in the San Garbiel Valley area.

#Popular shop

* db.map.aggregate([{"$match":{"shop":{"$exists":1}}}, {"$group":{"\_id":"$shop", "count":{"$sum":1}}}, {"$sort":{"count":­-1}}])



* SGV has a decent number of supermarket and convenience store. However, since I have been living in the area for more than 20 years, based on my observation, this number is smaller than the actual number. Therefore, I believe the data is incomplete.

Auditing the Data

Step 1: Import library and file. Parse through the file with ElementTree to find the counts of each element.

*#Import library*

**import** xml.etree.cElementTree **as** ET

**from** collections **import** defaultdict

**import** re

**import** pprint

*#Import file*

OSMFILE = "example.osm"

tags = {}

**for** event, elem **in** ET.iterparse(OSMFILE):

**if** elem.tag **in** tags:

tags[elem.tag] += 1

**else**:

tags[elem.tag] = 1

pprint.pprint(tags)

{'bounds': 1,

'member': 16372,

'meta': 1,

'nd': 320045,

'node': 249298,

'note': 1,

'osm': 1,

'relation': 561,

'tag': 220126,

'way': 36774}

Step 2: From lesson 6, we defined lower, lower\_colon, and problemchars. The code below is to show the counts of each of the definition below.

Lower = Strings containing lower case characters

Lower\_colon = Strings containing lower cases characters and a single colon within the string

Problemchars = characters that cannot be used within keys in MongoDB

lower = re.compile(r'^([a-z]|\_)\*$')

lower\_colon = re.compile(r'^([a-z]|\_)\*:([a-z]|\_)\*$')

problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

**def** key\_type(element, keys):

**if** element.tag **==** "tag":

**if** problemchars.search(element.attrib['k']):

keys['problemchars'] += 1

**elif** lower.search(element.attrib['k']):

keys['lower'] += 1

**elif** lower\_colon.search(element.attrib['k']):

keys['lower\_colon'] += 1

**else**:

keys['other'] += 1

**return** keys

**def** process\_map(OSMFILE):

keys = {"lower": 0, "lower\_colon": 0, "problemchars": 0, "other": 0}

**for** \_, element **in** ET.iterparse(OSMFILE):

keys = key\_type(element, keys)

**return** keys

keys = process\_map(OSMFILE)

pprint.pprint(keys)

{'lower': 116548, 'lower\_colon': 100586, 'other': 2990, 'problemchars': 2}

Step 3: Build an expression to match the token in a sting ending with a period and the expected clean street type.

street\_type\_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

expected\_street\_types = ["Avenue", "Boulevard", "Commons", "Court", "Drive", "Lane", "Parkway",

"Place", "Road", "Square", "Street", "Trail"]

map\_street\_types = \

{

"Ave" : "Avenue",

"BLVD" : "Boulevard",

"Blvd" : "Boulevard",

"Blvd." : "Boulevard",

"Cir" : "Circle",

"Dr" : "Drive",

"Ln" : "Lane",

"Pkwy" : "Parkway",

"Rd" : "Road",

"Rd." : "Road",

"St" : "Street",

"St." : "Street"

}

Step 4: The audit\_string function will take in the dictionary of the regex and street types in step 3 to match against that string and the list of expected street types. If there is a match and the match is not in our list, then it will add the match as a key to the dictionary and the string to the set.

def audit\_string(match\_set\_dict, string\_to\_audit, regex, expected\_matches):

m = regex.search(string\_to\_audit)

if m:

match\_string = m.group()

if match\_string not in expected\_matches:

match\_set\_dict[match\_string].add(string\_to\_audit)

Step 5: The audit function will do the parsing and auditing of the street names in our database.

def audit(osmfile, tag\_filter, regex, expected\_matches = []):

osm\_file = open(osmfile, "r")

match\_sets = defaultdict(set)

# iteratively parse

for event, elem in ET.iterparse(osm\_file, events=("start",)):

# iterate the tags within node or way

if elem.tag == "node" or elem.tag == "way":

for tag in elem.iter("tag"):

if tag\_filter(tag):

audit\_string(match\_sets, tag.attrib['v'], regex, expected\_matches)

return match\_sets

Step 6: The is\_street\_name function will determine if the element has “addr:street” attribute. Also, we can see a print of the audit output.

def is\_street\_name(elem):

return (elem.attrib['k'] == "addr:street")

street\_types = audit(OSMFILE, tag\_filter = is\_street\_name, regex = street\_type\_re,

expected\_matches = expected\_street\_types)

pprint.pprint(dict(street\_types))

{'106': set(['South Lake Ave #106']),

'21': set(['South Avenue 21']),

'3rd': set(['E. 3rd']),

'52': set(['Avenue 52']),

'53': set(['N Avenue 53']),

'56': set(['N Avenue 56']),

'61': set(['North Avenue 61']),

'64': set(['North Avenue 64']),

'Alley': set(['Martin Alley', 'Miller Alley']),

'Ave': set(['210 N Mayflower Ave',

'718 S Azusa Ave',

'850 S Baldwin Ave',

'Arcadia Ave',

'Bradshawe Ave',

'Camino Real Ave',

'Cesar E Chavez Ave',

'Clark Ave',

'Durfee Ave',

'East Naomi Ave',

'Eastern Ave',

'Evergreen Ave',

'Fair Oaks Ave',

'Fairview Ave',

'Fetterly Ave',

'Garfield Ave',

'Glendora Ave',

'Granada Ave',

'Harkness Ave',

'Holliston Ave',

'Indiana Ave',

'Lemon Ave',

'Lincoln Ave',

'Live Oak Ave',

'Longden Ave',

'Maine Ave',

'Michigan Ave',

'N Allen Ave',

'N Azusa Ave',

'N Raymond Ave',

'Naomi Ave',

'Nelson Ave',

'North Allen Ave',

'North Fair Oaks Ave',

'S Lake Ave',

'S. Atlantic Ave',

'Santa Ana Ave',

'Santa Anita Ave',

'South Catalina Ave',

'South Fair Oaks Ave',

'South Freemont Ave',

'Sw Quad Sr 134 / Sr 2 Ic Off E Wilson Ave',

'W. Clary Ave',

'W. Commonwealth Ave',

'Windsor Ave',

'Wistaria Ave']),

'Ave.': set(['Muscatel Ave.', 'North Fair Oaks Ave.', 'S. Lake Ave.']),

'Blvd': set(['Atlantic Blvd',

'Baldwin Park Blvd',

'Del Mar Blvd',

'East Colorado Blvd',

'East Washington Blvd',

'Hacienda Blvd',

'Ramona Blvd',

'Rosemead Blvd',

'San Gabriel Blvd',

'Valley Blvd',

'Washington Blvd',

'West California Blvd',

'Whittier Blvd']),

'Blvd.': set(['California Blvd.',

'East Colorado Blvd.',

'East Washington Blvd.',

'W. Foothill Blvd.',

'W. Valley Blvd.',

'Wilshire Blvd.']),

'Broadway': set(['East Broadway', 'North Broadway']),

'Clary': set(['W. Clary']),

'Crest': set(['Sienna Crest']),

'Dr': set(['Carter Dr',

'Floral Dr',

'Huntington Dr',

'Pepperglen Dr',

'Rosilyn Dr',

'Woolwine Dr']),

'Ducommun': set(['Ducommun']),

'E': set(['W. Main St. Suite E']),

'Grande': set(['Potrero Grande']),

'Hwy': set(['Arrow Hwy']),

'Ln': set(['La Vida Ln']),

'North': set(['Crossroads Parkway North']),

'Pkwy': set(['South Arroyo Pkwy']),

'Plaza': set(['Gateway Plaza']),

'Rd': set(['437 W San Bernardino Rd',

'Don Julian Rd',

'Duarte Rd',

'Peck Rd',

'Rivergrade Rd',

'Sharon Rd']),

'Rd.': set(['Amar Rd.']),

'Rosemead': set(['N. Rosemead']),

'ST': set(['W 6TH ST']),

'South': set(['Huntington Drive South']),

'St': set(['1st St',

'3rd St',

'Alameda St',

'Blanchard St',

'Callita St',

'Comly St',

'Dozier St',

'E Green St',

'E. 1st St',

'E. Green St',

'Eugene St',

'Fisher St',

'Folsom St',

'Gabriel Garcia Marquez St',

'Hammel St',

'Indiana St',

'Main St',

'Mesquit St',

'Mission St',

'N Figueroa St',

'New York St',

'North Spring St',

'S Alameda St',

'Stafford St',

'Val St',

'W Badillo St']),

'St.': set(['Center St.',

'Commercial St.',

'Ducommun St.',

'E. 1st St.',

'E. 3rd St.',

'E. Commercial St.',

'E. Mendocino St.',

'E. Temple St.',

'Jackson St.',

'Judge John Aiso St.',

'N. Alameda St.',

'N. Gary St.',

'N. Los Angeles St.',

'South Almansor St.',

'Temple St.']),

'Windsor': set(['2122 N Windsor']),

'madera': set(['madera'])}

Step 7: The update function will replace the abbreviated street types. The string to update function will take a string to update, mapping the dictionary, and a regex to search.

def update(string\_to\_update, mapping, regex):

m = regex.search(string\_to\_update)

if m:

match = m.group()

if match in mapping:

string\_to\_update = re.sub(regex, mapping[match], string\_to\_update)

return string\_to\_update

Step 8: Take the keys from the map to create a string joined by “|”. This will cause the regex to search for any of the keys to match the first it finds.

bad\_street\_name = "|".join(map\_street\_types.keys()).replace('.', '')

street\_type\_updater\_re = re.compile(r'\b(' + bad\_streets + r')\b\.?', re.IGNORECASE)

Step 9: Traverse the street\_type dictionary to see the abbreviations to clean representations.

for street\_type, ways in street\_types.iteritems():

if street\_type in map\_street\_types:

for name in ways:

new\_name = update(name, map\_street\_types, street\_type\_updater\_re)

print name, "=>", new\_name

La Vida Ln => La Vida Lane

Center St. => Center Street

Jackson St. => Jackson Street

Judge John Aiso St. => Judge John Aiso Street

N. Alameda St. => N. Alameda Street

E. Temple St. => E. Temple Street

South Almansor St. => South Almansor Street

E. 1st St. => E. 1st Street

Commercial St. => Commercial Street

E. 3rd St. => E. 3rd Street

E. Commercial St. => E. Commercial Street

Temple St. => Temple Street

N. Gary St. => N. Gary Street

N. Los Angeles St. => N. Los Angeles Street

E. Mendocino St. => E. Mendocino Street

Ducommun St. => Ducommun Street

Duarte Rd => Duarte Road

Don Julian Rd => Don Julian Road

437 W San Bernardino Rd => 437 W San Bernardino Road

Sharon Rd => Sharon Road

Peck Rd => Peck Road

Rivergrade Rd => Rivergrade Road

Amar Rd. => Amar Road

Huntington Dr => Huntington Drive

Floral Dr => Floral Drive

Carter Dr => Carter Drive

Rosilyn Dr => Rosilyn Drive

Woolwine Dr => Woolwine Drive

Pepperglen Dr => Pepperglen Drive

Folsom St => Folsom Street

E. 1st St => E. 1st Street

Indiana St => Indiana Street

Blanchard St => Blanchard Street

E. Green St => E. Green Street

North Spring St => North Spring Street

Callita St => Callita Street

New York St => New York Street

Mission St => Mission Street

1st St => 1st Street

N Figueroa St => N Figueroa Street

Dozier St => Dozier Street

Main St => Main Street

W Badillo St => W Badillo Street

3rd St => 3rd Street

Eugene St => Eugene Street

Mesquit St => Mesquit Street

Hammel St => Hammel Street

Gabriel Garcia Marquez St => Gabriel Garcia Marquez Street

Comly St => Comly Street

Val St => Val Street

Stafford St => Stafford Street

Fisher St => Fisher Street

E Green St => E Green Street

S Alameda St => S Alameda Street

Alameda St => Alameda Street

Michigan Ave => Michigan Avenue

Fairview Ave => Fairview Avenue

Windsor Ave => Windsor Avenue

Garfield Ave => Garfield Avenue

Fetterly Ave => Fetterly Avenue

Holliston Ave => Holliston Avenue

210 N Mayflower Ave => 210 N Mayflower Avenue

Lincoln Ave => Lincoln Avenue

Bradshawe Ave => Bradshawe Avenue

South Catalina Ave => South Catalina Avenue

Nelson Ave => Nelson Avenue

Arcadia Ave => Arcadia Avenue

Clark Ave => Clark Avenue

Fair Oaks Ave => Fair Oaks Avenue

Indiana Ave => Indiana Avenue

850 S Baldwin Ave => 850 S Baldwin Avenue

Longden Ave => Longden Avenue

Granada Ave => Granada Avenue

Durfee Ave => Durfee Avenue

Wistaria Ave => Wistaria Avenue

Santa Ana Ave => Santa Ana Avenue

Naomi Ave => Naomi Avenue

Eastern Ave => Eastern Avenue

Cesar E Chavez Ave => Cesar E Chavez Avenue

Live Oak Ave => Live Oak Avenue

Glendora Ave => Glendora Avenue

Lemon Ave => Lemon Avenue

Santa Anita Ave => Santa Anita Avenue

N Raymond Ave => N Raymond Avenue

W. Clary Ave => W. Clary Avenue

S Lake Ave => S Lake Avenue

East Naomi Ave => East Naomi Avenue

N Allen Ave => N Allen Avenue

N Azusa Ave => N Azusa Avenue

Maine Ave => Maine Avenue

North Allen Ave => North Allen Avenue

S. Atlantic Ave => S. Atlantic Avenue

Sw Quad Sr 134 / Sr 2 Ic Off E Wilson Ave => Sw Quad Sr 134 / Sr 2 Ic Off E Wilson Avenue

South Fair Oaks Ave => South Fair Oaks Avenue

718 S Azusa Ave => 718 S Azusa Avenue

W. Commonwealth Ave => W. Commonwealth Avenue

North Fair Oaks Ave => North Fair Oaks Avenue

Evergreen Ave => Evergreen Avenue

Harkness Ave => Harkness Avenue

Camino Real Ave => Camino Real Avenue

South Freemont Ave => South Freemont Avenue

South Arroyo Pkwy => South Arroyo Parkway

W. Valley Blvd. => W. Valley Boulevard

W. Foothill Blvd. => W. Foothill Boulevard

California Blvd. => California Boulevard

East Colorado Blvd. => East Colorado Boulevard

East Washington Blvd. => East Washington Boulevard

Wilshire Blvd. => Wilshire Boulevard

Baldwin Park Blvd => Baldwin Park Boulevard

West California Blvd => West California Boulevard

East Colorado Blvd => East Colorado Boulevard

Whittier Blvd => Whittier Boulevard

Del Mar Blvd => Del Mar Boulevard

San Gabriel Blvd => San Gabriel Boulevard

East Washington Blvd => East Washington Boulevard

Ramona Blvd => Ramona Boulevard

Valley Blvd => Valley Boulevard

Washington Blvd => Washington Boulevard

Hacienda Blvd => Hacienda Boulevard

Rosemead Blvd => Rosemead Boulevard

Atlantic Blvd => Atlantic Boulevard

Conclusion

Based on the analysis of this dataset, I can conclude that the dataset is incomplete and still need a lot of work. In my “audit the data” exercise, I cleaned up the abbreviation in the address names so they are more uniform. However, there are much more data that needs to clean up, such as the postal code, cardinal directions of the address, and amenity names. Beyond the scope of this course, we can analyze the inputs from each users and see what kind of data format/error they are tend to create so we can better produce an uniform dataset.

I am glad that openstreetmap open their data source and allow users to make updates to improve the dataset. This exercise taught me how to do queries and analysis big data using MongoDB and pylon. I believe the users that are making update to openstreetmap also enjoy learning about the dataset and improving their skills in data wrangling.