Daniel Nitiutomo

Project Summary

Name

Daniel Nitiutomo

Udacity Email

dnitiutomo@gmail.com

Area

URL – http://www.openstreetmap.org/export#map=11/40.7322/-73.9747

NOTE: The above is an approximation. I used the MapZen Metro Extracts to download the New York City OSM xml file. Can be found here: https://mapzen.com/metro-extracts/

I used New York City because it is where I have lived for the past 7 years and I thought it would be most interesting to see data about a city I knew.

Resources Used

- Stack Overflow
  - Contained explanations of python solutions for common problems such as manipulating strings and sorting dictionaries.
  - http://stackoverflow.com/questions/613183/sort-a-python-dictionary-by-value
- http://docs.mongodb.org/
  - Used the MongoDB documentation website to understand how to use the MongoDB python driver and for common commands involving the database management through the command line.
- https://docs.python.org/2/library/xml.etree.elementtree.html
  - This documentation allowed me to understand how to properly use the Element Tree python package to analyze the OSM XML file.
- http://effbot.org/zone/element-iterparse.htm
  - This web page described how to properly dispose of Element Tree iterparse objects to prevent the parsing script from using too much memory.
- wiki.openstreetmap.org
  - Provided descriptions of all the fields and tags that are used within the OSM XML file.
- Udacity Data Wrangling with MongoDB Course
  - The course gave me the code from which I created my OSM parsing scripts and my MongoDB querying script.

Statement of Integrity

"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."

Problems Encountered

After analyzing the OSM XML file, I encountered multiple formatting decisions and inconsistencies that needed to be handled by my data parsing script before being entered into the database.

Nested Keys

The first issue that I encountered was the way the keys were structured. Multiple keys existed that had nested keys defined by a ":" character. There could be as many as 3 separate items populating a single key. Examples include:

- "change:lanes:backward"
- "source:hgv:forward"
- "motor_vehicle:conditional"

I had to account for this using a generic solution that would be able to create new nested dictionaries on the fly and add new entries to those nested dictionaries as they appeared. I did this by separating the keys using the ":" character and initializing dictionaries based on the nested structure if they did not exist.

City Entries used for Neighborhood, Borough, and/or City

The 'city' field was used inconsistently throughout the OSM data set. It was used to describe the City, Borough, or Neighborhood in which the node was located within. In order to make the data set consistent, I created a CSV file that contained all of the mappings between New York City boroughs and neighborhoods. I imported these mappings as a dataframe and checked whether the city field matched any neighborhood or borough. If it did, then I inputted "New York" as the city and created separate fields to enter the borough and neighborhood.

Multiple Names for Address Items

For the street, city, and state fields, there were multiple ways of describing the same field. For example, the city "New York" was also described as "NY", "nyc", "New York NY", and "Brooklyn, NY, NY." Streets and states were named in full and abbreviated form as well. I made the names consistent by updating all the strings so that they used full name form.

Post Codes

Post codes were defined using the normal ('XXXXX') and extended format ('XXXXX-XXXX'). Also, there were a number of post codes that were defined with the city and state included. I used regex in order to identify valid post codes. I also separated the primary 5-digit post code and the extended 4-digit number into two different address fields.

I also used post codes to determine in which borough the address was located within. I created a dataframe with the zip code to borough mappings and then analyzed the zip code to properly populate the borough field.

Data Set Area Range

By analyzing the cities and states within the data set, I noticed that some cities were located over 30 miles outside of New York in New Jersey and Connecticut. Therefore, this data set should have been named the "Greater New York City Area" rather than simply "New York."

Query for States:

```
aggregate([{'$group':{'_id':'$address.state, 'count':{'$sum':1}}},
{'$sort':{'count':1}}])
```

Results (Item, Count):

```
{None : 9760690, NY : 4355, NJ : 1539, CT : 89, VA : 1, ON : 1}
```

It is obvious that there are multiple states included within the analysis. When I viewed the outlier entries, VA and ON, I concluded that they must be errors. The 'VA' entry was for a Farmers Market in Middletown, Virginia. There most likely was a mistake because there were multiple entries for the town Middletown, New Jersey; which happens to be a suburb outside of New York City. The 'ON' entry actually corresponded to a McDonald's in Brooklyn, NY, NY.

Unknown Metrics for Numeric Fields

Numeric fields within the data set would sometimes include the metrics they were expressed in and would sometimes exclude them. The first task was to clean the values and separate the number from the metric if the metric existed. I converted the number to a float and added it to the node object using the corresponding key. Then I added the metric as an additional field to describe the number.

Even after the values were cleaned, however, there were many numbers that had no accompanying metric. This makes the data difficult to compare across all entries.

Inconsistent Use of Fields

Many of the data set's fields were used inconsistently between entries. For example, the 'building' field had a nested attribute 'levels.' This 'levels' field was sometimes used to describe the number of levels within a building and would also be used to describe the grade levels that a school accommodated. I did not attempt to fix these inconsistencies as these were a case by case basis for which a generic solution would be very difficult and would not add much value.

Overview of the Data

File Sizes

```
new-york_new-york.osm …… 2.127 GB

new_york.json …….. 2.392 GB
```

Number of Documents

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

9766675
```

Number of Nodes

```
> db.nyc.find({"element_type":"node"}).count()

8383738
```

Number of Ways

```
> db.nyc.find({"element_type":"way"}).count()

1382937
```

Number of Unique Users

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

2598
```

Top 1 Contributing User

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

{ "_id" : "Rub21_nycbuildings", "count" : 4903163 }
```

Number of Users Appearing Only Once

```
>db.nyc.aggregate([{'$group':{'_id':'$created.user','count':{'$sum':1}}},{'$group':{'_id':'$count','num_users':{'$sum':1}}},{'$sort':{'_id':1}},{'$limit':1}])

{ "_id" : 1, "num_users" : 601 }
```

Additional Ideas

Most Prevalent Names within the Data Set

After conducting a simple query for the most common names within the data set, I found there were a common establishments that recurred other than street names, highways, and transportation.

- Starbucks with 128 occurrences
- McDonald's with 115 occurrences
- Chase with 97 occurrences
- Duane Reade with 76 occurrences
- Dunkin Donuts with 70 occurrences
- Bank of America with 69 occurrences
- TD Bank with 64 occurrences
- Subway with 62 occurrences

Even though there was not a plethora of data to go off of, it seems as though New Yorkers truly love a good cup of Starbucks and opt for the fast food discount. From personal experience, I can confirm the stores, banks, and shops above are everywhere.

Comparing the Boroughs of NYC

I wanted to compare differences between the NYC boroughs since many New Yorkers identify with the borough that they live or grew up in. I conducted several queries using the python MongoDB drivers. Below are some of the queries for each borough.

Top 5 Amenities per Borough

Manhattan	**Brooklyn**	**Queens**	**Bronx**	**Staten Island**
restaurant : 434	restaurant : 253	place_of_worship : 229	place_of_worship : 145	place_of_worship : 20
embassy : 176	cafe : 66	restaurant : 91	school : 61	restaurant : 14
place_of_worship : 154	bar : 54	school : 66	library : 35	library : 13
cafe : 127	library : 38	bank : 37	hospital : 21	post_office : 6
fast_food : 98	fast_food : 31	fast_food : 31	restaurant : 16	fire_station : 3

Manhattan, Brooklyn, and Queens are all renowned for their food, so it is no surprise that restaurants would occupy the first or second spot on the list. What is surprising to me is how prevalent places of worship are in each borough, except for Brooklyn. For Queens, Bronx, and Staten Island, places of worship are the top amenity by a significant amount. I would also say that this data indicates that Queens, the Bronx and Staten Island are more residential areas as the percentage of schools, libraries, and religious institutions are much higher.

Top 5 Cuisines per Borough

Manhattan	**Brooklyn**	**Queens**	**Bronx**	**Staten Island**
coffee_shop : 57	coffee_shop : 23	burger : 14	italian : 5	sri_lankan : 3
italian : 46	american : 18	american : 10	burger : 3	american : 3
burger : 42	italian : 16	coffee_shop : 9	sandwich : 2	ice_cream : 2
pizza : 39	mexican : 15	pizza : 8	doughnuts : 1	italian : 2
mexican : 33	pizza : 15	mexican : 7	steak_house : 1	burger : 1

There was not much data here, but this does suggest that Manhattan and Brooklyn folk love their coffee shops. This is also consistent with the fact that Starbucks is the most prevalent chain in New York City.

Example Query for Borough Data

```
pipeline.append({'$match':{'address.borough':'Brooklyn'}})
pipeline.append({'$group':{'_id':'$cuisine','count':{'$sum':1}}})
pipeline.append({'$sort':{'count':-1}})
```

Additional Data Exploration using MongoDB Queries

Top 10 Cuisines

```
>
db.nyc.aggregate([{'$group':{'_id':'$cuisine','count':{'$sum':1}}},{'$sort':{
'count':-1}},{'$limit':11}])
```

```
{ "_id" : null, "count" : 9764729 }
{ "_id" : "burger", "count" : 216 }
{ "_id" : "coffee_shop", "count" : 211 }
{ "_id" : "italian", "count" : 151 }
{ "_id" : "pizza", "count" : 150 }
{ "_id" : "american", "count" : 148 }
{ "_id" : "mexican", "count" : 119 }
{ "_id" : "sandwich", "count" : 82 }
{ "_id" : "chinese", "count" : 74 }
{ "_id" : "donut", "count" : 46 }
{ "_id" : "japanese", "count" : 44 }
```

Top 10 Shop Types

```
>
db.nyc.aggregate([{'$group':{'_id':'$shop','count':{'$sum':1}}},{'$sort':{'co
unt':-1}},{'$limit':11}])
```

```
{ "_id" : null, "count" : 9763407 }
{ "_id" : "supermarket", "count" : 461 }
{ "_id" : "clothes", "count" : 328 }
{ "_id" : "convenience", "count" : 327 }
{ "_id" : "alcohol", "count" : 118 }
{ "_id" : "department_store", "count" : 114 }
{ "_id" : "car_repair", "count" : 110 }
{ "_id" : "bakery", "count" : 100 }
{ "_id" : "hairdresser", "count" : 97 }
{ "_id" : "yes", "count" : 90 }
{ "_id" : "mall", "count" : 88 }
```