☐ **README.md**

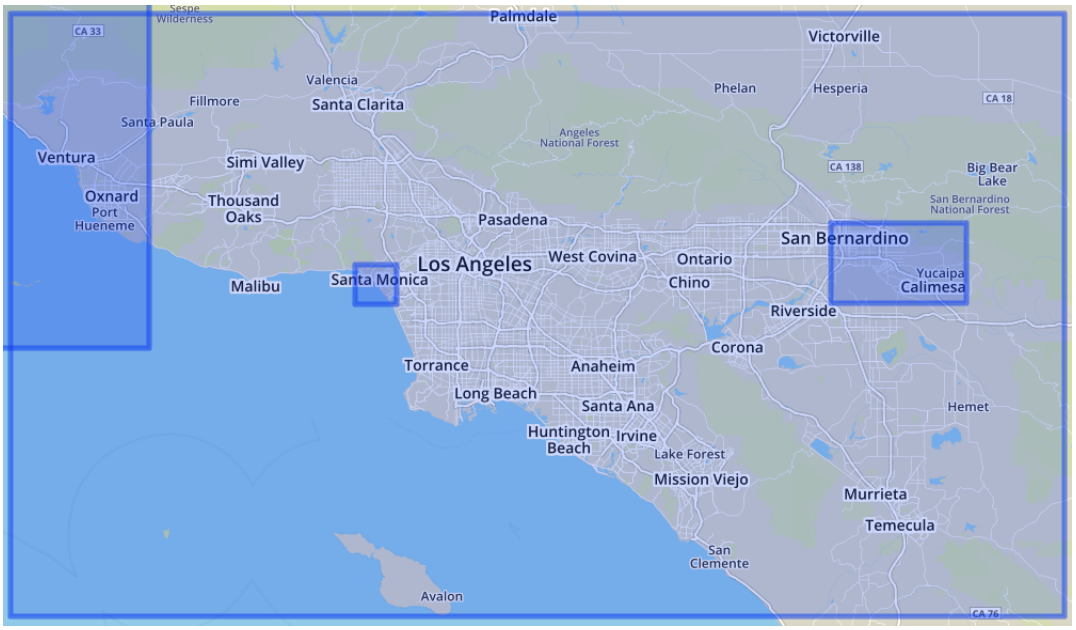
OSM Data Cleanup Project Notes

Submission Details

1. This README is my project writup.
2. Project 6 solutions can be found in Udacity Data Wrangling Lesson 6.ipynb
3. See The Map Data section for more info on what OSM data I used and why I chose it.
4. A thinned version of the data I used is located in this repository, https://github.com/dfmcmurray/udacity-data-wrangling-project/blob/master/los-angeles_california_thinned.osm.
5. My references can also be found in this repository, https://github.com/dfmcmurray/udacity-data-wrangling-project/blob/master/REFERENCES.txt.

The Map Data

Downloaded OSM Data from https://mapzen.com/data/metro-extracts for a large area surrounding the Los Angeles metro area, including my current town of residence, Pasadena. The bounding box for the map data I used is the largest blue square that's lined up with the screen shot. The size of the data file that I worked with was 1.21 GB. For the project submission, I included every 150th top-level node, bringing the data file down to 8.2 MB.



Data Model

After a quick audit, I came up with an initial data model for the OSM data. The following ways are how I planned to clean the OSM data.

Problem Characters

Any key with any "problem characters", defined by this regular expression `[=\+/&<>;\'"\?%#$@\,\. \t\r\n]` , will be removed.

address

Anything starting with "addr:" will be converted to an `address` object. Also, convert street abbreviations to full type (e.g., "Blvd" to "Boulevard").

```
-Ex:
    "address": {
        "housenumber": 5158,
        "street": "North Lincoln Avenue”,
        ...
    }
```

is_in

The `is_in` property is generated in the same what the `address` object is.

phone

The phone number data comes in a variety of formats. To clean them up, phone numbers will be stripped of non-numeric characters and broken up into their three sections, the area code, the three digit part, and the four digit part, or, by their more appropriately technical names, npa, nxx, and xxx respectively (https://en.wikipedia.org/wiki/North_American_Numbering_Plan#Numbering_system).

```
-Ex.
    “phone”: {
        “npa”: “213”,
        “nxx”: “555”,
        “xxxx”: “5555”
    }
```

open_hours

The open hours data will be converted into a dict with keys being days of week and values being arrays of 2-element tuples, where the first element is an opening time and the second element is a closing time.

```
-Ex.
    “open_hours”: {
        “Monday”: [(“9:00”, “14:00”), (“17:00”, “22:00”)]
        “Tuesday”: [(“9:00”, “14:00”), (“17:00”, “22:00”)]
        “Wednesday”: [(“9:00”, “14:00”), (“17:00”, “22:00”)]
        “Thursday”: [(“9:00”, “14:00”), (“17:00”, “22:00”)]
        “Friday”: [(“9:00”, “14:00”), (“17:00”, “22:00”)]
        “Saturday”: [(“9:00”, “22:00”)]
        “Sunday”: [(“9:00”, “22:00”)]
    }
```

Difficulties with the Data

Street Name Type

In the data model above I decided to replace all street name abbreviations with their full names. This proved a little harder that I originally thought. My first (naïve) approach just took the last word of the street field. This resulted in many incorrect selections, since the street type didn’t always come last. So I went for a more sophisticated approach, using the natural language address parsing module `usaddress` to suss out the “StreetNamePostType” from each street field, which worked much better.

Open Hours

Unfortunately, I ended up abandoning the above data model for `open_hours` and leaving the data unchanged. This model proved to be too difficult to encode due to the wide range of formats for this field (a real testament to the difficulty of using human entered data). I even tried using a natural language time parser (parsedatetime), but it wasn’t able to consistently get the correct times. Here are some examples of the data:

```
-24/7
-07:00-22:00
-May 15-Nov 16
```

```
-Mo-Fr 7:00-21:00; Sa-Su 7:00-21:00
-Monday:  Closed Tues-Sat: 11:00AM-2:00PM;5:00-8:00PM Sunday :   11:00AM-2:00PM
```

Exploring the Data

After cleaning the data and importing it into MongoDB, I ran the following python snippets (using pymongo commands) on the two mongo collections `nodes` and `ways`.

Total Elements

```
nodeCount = nodes.count()
wayCount = ways.count()
print "Total Number of Nodes:\t\t", nodeCount
print "Total Number of Ways:\t\t", wayCount
print "Total Number of All Documents:\t", nodeCount + wayCount
```

Output:

```
Total Number of Nodes:      5247813
Total Number of Ways:       562383
Total Number of All Documents:  5810196
```

Unique Users

```
def uniqueUsers(collection):
    return set(collection.distinct("created.user"))

uniqueUsersInNodes = uniqueUsers(nodes)
uniqueUsersInWays = uniqueUsers(ways)
print "Unique Users in Nodes:\t\t", len(uniqueUsersInNodes)
print "Unique Users in Ways:\t\t", len(uniqueUsersInWays)
print "Unique Users in All Documents:\t", len(uniqueUsersInNodes | uniqueUsersInWays)
```

Output:

```
Unique Users in Nodes:      2628
Unique Users in Ways:       1975
Unique Users in All Documents:  2871
```

Top Controbutors

```
def topContributingUsers(limit, *collections):
    if limit <= 0:
        return None
    contributors = {}
    for collection in collections:
        for contributor in collection.aggregate([{"$group":{"_id":"$created.user", "cou
            contributionCount = contributors.get(contributor["_id"], 0) + contributor["
            contributors[contributor["_id"]] = contributionCount
    return heapq.nlargest(limit, contributors.items(), lambda pair: pair[1])

topContributorLimit = 10
topContributorsInNodes = topContributingUsers(topContributorLimit, nodes)
topContributorsInWays = topContributingUsers(topContributorLimit, ways)
topContributorsInAllDocuments = topContributingUsers(topContributorLimit, nodes, ways)
print "Top {} Contributors in Nodes:".format(topContributorLimit)
print "".join(["{:<30}{:<10}({:.2%})\n".format(username, contributionCount, float(contr
print "Top {} Contributors in Ways:".format(topContributorLimitPerList)
print "".join(["{:<30}{:<10}({:.2%})\n".format(username, contributionCount, float(contr
print "Top {} Contributors in All Documents:".format(topContributorLimitPerList)
print "".join(["{:<30}{:<10}({:.2%})\n".format(username, contributionCount, float(contr
```

Output:

```
Top 10 Contributors in Nodes:
woodpeck_fixbot               546730    (10.42%)
The Temecula Mapper           452435    (8.62%)
AM909                         429283    (8.18%)
nmixter                       329645    (6.28%)
Brian@Brea                    206737    (3.94%)
Aaron Lidman                  154956    (2.95%)
SJFriedl                      137614    (2.62%)
Jon Schleuss                  132206    (2.52%)
jerjozwik                     130263    (2.48%)
mattmaxon                     116291    (2.22%)

Top 10 Contributors in Ways:
balrog-kun                    62022     (11.03%)
The Temecula Mapper           39083     (6.95%)
AM909                         33124     (5.89%)
Aaron Lidman                  26817     (4.77%)
Brian@Brea                    26430     (4.70%)
SJFriedl                      17471     (3.11%)
NE2                           11960     (2.13%)
Jon Schleuss                  10677     (1.90%)
DaveHansenTiger               10607     (1.89%)
jerjozwik                     10292     (1.83%)

Top 10 Contributors in All Documents:
woodpeck_fixbot               546732    (9.41%)
The Temecula Mapper           491518    (8.46%)
AM909                         462407    (7.96%)
nmixter                       330838    (5.69%)
Brian@Brea                    233167    (4.01%)
Aaron Lidman                  181773    (3.13%)
SJFriedl                      155085    (2.67%)
Jon Schleuss                  142883    (2.46%)
jerjozwik                     140555    (2.42%)
mattmaxon                     121211    (2.09%)
```

Top Amenities

```
def topAmenities(limit, collection):
    return collection.aggregate([{"$match":{"amenity":{"$exists":1}}}, {"$group":{"_id"

def keyToTitle(key):
    return key.replace("_", " ").title()

topAmenitiesLimit = 10
topAmenitiesList = topAmenities(topAmenitiesLimit, nodes)
print "Top {} Amenity Types:".format(topAmenitiesLimit)
print "".join(["{:<30}{:<10}\n".format(keyToTitle(amenity["_id"]), amenity["count"]) fo
```

Output:

```
Top 10 Amenity Types:
Place Of Worship              3788
School                        3123
Restaurant                    1725
Fast Food                     1254
Fuel                          745
Cafe                          576
Fountain                      493
Parking                       490
Toilets                       483
Drinking Water                458
```

Top Cuisines

```
def topCuisines(limit, collection):
    return collection.aggregate([{"$match":{"cuisine":{"$exists":1}, "amenity":"restaur

def keyToTitle(key):
    return key.replace("_", " ").title()

topCuisinesLimit = 15
topCuisinesList = topCuisines(topCuisinesLimit, nodes)
print "Top {} Cuisines:".format(topCuisinesLimit)
print "".join(["{:<30}{:<10}\n".format(keyToTitle(cuisine["_id"]), cuisine["count"]) fo
```

Output:

```
Top 15 Cuisines:
American                      155
Mexican                       145
Pizza                         84
Italian                       67
Chinese                       57
Japanese                      47
Thai                          41
Burger                        40
Sushi                         39
Sandwich                      33
Seafood                       24
Steak House                   17
Indian                        16
Asian                         15
Regional                      15
```

Top Restaurants

```
def topRestaurants(limit, collection):
    return collection.aggregate([{"$match":{"name":{"$exists":1}, "amenity":"restaurant

def keyToTitle(key):
    return key.replace("_", " ").title()

topRestaurantsLimit = 20
topRestaurantsList = topRestaurants(topRestaurantsLimit, nodes)
print "Top {} Restaurants:".format(topRestaurantsLimit)
print "".join(["{:<30}{:<10}({})\n".format(restaurant["_id"], restaurant["count"], ", "
```

Output:

```
Top 20 Restaurants:
Denny's                       23        (American, Diner)
Subway                        17        (Sandwich)
IHOP                          15        (Breakfast, American, Pancake)
Panda Express                 10        (American, Chinese)
Chipotle                      9         (Mexican)
Pizza Hut                     9         (Pizza)
California Pizza Kitchen      9         (Pizza, Italian)
Sizzler                       7         (American, Steak)
Red Robin                     7         (Burger, American)
Ruby's Diner                  7         (American)
Cheesecake Factory            7         (American)
Rubio's                       6         (Mexican)
Chipotle Mexican Grill        5         (Mexican)
Islands                       5         (Burger, American)
Round Table Pizza             5         (Pizza)
Corner Bakery                 5         (Chinese)
Carrows                       5         (American)
Olive Garden                  5         (Italian)
Chili's                       4         ()
BJ's                          4         (American)
```

Concusion and Other Ideas

I would suggest that the OSM database start combining user generated data with automated data retrieval methods. There is a upcoming federal government effort to install sensors to make our cities smarter (https://www.whitehouse.gov/the-press-office/2015/09/14/fact-sheet-administration-announces-new-smart-cities-initiative-help). The data from these sensors could be augment the OSM data which is more structural to include more information about how people move through and use the cities' infrastructure and amenities. However, since this project is in the very early stages, it is not yet clear if the sensor data would be a good match for the user-generated OSM data. At this point, it's just a good idea to keep an eye on the smart city project's development.

I would also suggest an improvement to the OSM's website to spark more interest in the datasets. Taking another cue from the U.S. government, http://www.data.gov/ has a great landing page for a data website. A visitor is immediately presented with categories to dig into, which can spark excitement and insight when dealing with the data. If a visitor could submit basic queries to gather statistics like the ones I presented in the last section, then the data would be much more accessible to the non-technical, and it would open up OSM's user base to that wider population. The key to keeping a user-generated dataset thriving is to have as many people invested as possible, so this change to the website's landing page would be a big step toward expanding the user base.

The sensor integration approach and the widening of the user base approach would both serve this dataset well by expanding the dataset even more. This data set, which already huge, can't be complete. (I mean, there are only 2 "In N Out"s in the dataset, for crying out lout!) This point is clearer when you realize that Los Angeles County has more people in it than the 11 smallest U.S. states combined (10,116,705 versus 9.870,265; http://dadaviz.com/s/population-extremes). However, with that many people, if the word about the OSM database were spread to even a small portion of them who would participate, it wouldn't take long to get a much more complete picture.