Data Set overview:

Stats as of 8-1-16

- Size of osm uncompressed: 433.8 MB
- Number of Unique Users: 969
- Number of Nodes: 1930900
- Number of Ways: 308199

Challenges Encountered:

- Address parsing. Many nodes have both an `address` key/ value pair and `addr:XYZ` fields for a second copy of the address. I added the `is_address` and `is_street` helpers and branch address parsing/ cleaning on the return value of these helpers.
- Dirty address data: There was a fair number of ways that addresses were written out in their respective fields. I first did some auditing to get a feel for the data, and then acculmulated some cases (`test_data` in `utils.py`). From there I wrote the method `clean_address` to parse and standardize the address data. Finally, the two aspects of data that I standardized were the street suffixes (see `mapping` in `utils.py`) and the state value (`Massachusetts` instead of `Ma`, `Mass`, etc).
- Often the top result for group & sum aggregation queries returns `None`. This makes sense as not all tag types will have a value. However, this should be cleaned from the data set.

Resources:

MongoDB and PyMongo docs and Stack Overflow.

Additional Stats:

There is quite a bit of dirty data from these queries. For instance, there are at least four names for Starbucks ("Starbucks", "Starbuck's Coffee", "Starbucks Coffee", and "Starbucks (SMG)"). This would be a good area to do further data cleaning.

Important Note: I went through the returned cursors and added the variants together for the totals in this section. However, I didn't clean the data so the results here will appear slightly off.

- **Coffee shops:**

```
db.boston_massachusetts.aggregate([{$match: { "amenity" : "cafe", "cuisine": "coffee_shop" }}, {$group: {_id: "$name", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}])
```

This shows that Starbucks is the most numerous coffee shop with a total of 29 locations, followed by Dunkin's with 10. (Dunkin's tends to be classified as a cafe where it has 49 locations).

```
{ "_id" : "Starbucks", "count" : 25 }
{ "_id" : "Dunkin' Donuts", "count" : 10 }
{ "_id" : "Starbuck's Coffee", "count" : 2 }
{ "_id" : "Ula Cafe", "count" : 1 }
{ "_id" : "Ames Street Deli", "count" : 1 }

etc ...
```

- **Cafes:**

```
db.boston_massachusetts.aggregate([{$match: { "amenity" : "cafe" }}, {$group: {_id:
"$name", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}])
```

A more general query than above, but I think this gives a better view of the data. The same issues with data cleaning exist, and after the chains (Starbucks, Dunkin's, Au Bon Pain) it's mostly independent locations with a single or perhaps two locations.

```
{ "_id" : "Dunkin' Donuts", "count" : 46 }
{ "_id" : "Starbucks", "count" : 34 }
{ "_id" : "Au Bon Pain", "count" : 6 }
{ "_id" : "Starbucks Coffee", "count" : 5 }
{ "_id" : "Peet's Coffee", "count" : 3 }

etc ...
```

Universities / Colleges:

```
db.boston_massachusetts.aggregate([{$match: { "amenity" : {$in: ["university",
"college"]} }}, {$group: {_id: "$name", "count": {"$sum": 1}}}, {"$sort": {"count":
-1}}])
```

Here Boston University has the most data points with 41. However, a challenge with this query is that it doesn't take into account names that don't directly conform (Eg: "Harvard" has 5 data points, but doesn't include "Harvard Medical School", or other related locations like "Eliot House")

```
{ "_id" : "Boston University", "count" : 41 }
{ "_id" : "Massachusetts Institute of Technology", "count" : 10 }
{ "_id" : "Suffolk University", "count" : 8 }
{ "_id" : "Emerson College", "count" : 7 }
{ "_id" : "Berklee College of Music", "count" : 7 }

etc ...
```

- **Number of bookstores:** 25

```
db.boston_massachusetts.find({"shop": "books"}).count()
```

- **Banks:**

```
db.boston_massachusetts.aggregate([{$match: { "amenity" : "bank" }}, {$group: {_id:
"$name", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}])
```

Bank of America comes in with 14 locations followed by Citizens Bank with 11. Note the same issues with cleaning bank names exist.

```
{ "_id" : "Bank of America", "count" : 13 }
{ "_id" : "Citizens Bank", "count" : 10 }
{ "_id" : null, "count" : 6 }
{ "_id" : "TD Bank", "count" : 6 }
{ "_id" : "Eastern Bank", "count" : 6 }

etc ...
```

- **Convenience shops:**

```
db.boston_massachusetts.aggregate([{$match: { "shop" : "convenience" }}, {$group:
{_id: "$name", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}])
```

Here 7-Eleven (or 7/11) has the most locations at 14.

```
{ "_id" : "7-Eleven", "count" : 12 }
{ "_id" : null, "count" : 8 }
{ "_id" : "Tedeschi Food Shops", "count" : 5 }
{ "_id" : "City Convenience", "count" : 2 }
{ "_id" : "7/11", "count" : 2 }

etc ...
```

Ideas for further work:

- The `source` tag seems like a good candidate for further data cleaning and analysis. It looks like the majority of the sources are from MASS GIS and MASS DOT. However, the data is rather dirty, and there are a number of cases where people, search engines, or other nonstandard values are listed. Additionally, there's a number of links to the source, but many are broken. These could be parsed, validated, and linked to the data that contributes to a confidence index in the related data points. Additionally, the source data could be listed with the node/ way data for further investigation by users.

A couple challanges to this could be that this value doesn't exist for every data point in the data set, that links are broken, that the sources come from widely varying sources (Eg: a GIS database, pdf, or "local knowledge").

- The `pos` values in the cleaned data set could be correlated with the amenity/ tag type data to show clustering or densities for the selected tag type. This could displayed as a heat map or used as input to make prediction about where to to go for a given tag type.

- Clean the dirty data mentioned in the additional Stats section. This would make the queries more accurate.
- Clean phone numbers. There's a bit of variance in how phone numbers are present. Since we have a gold standard (a phone book) and precise way of representing phone numbers, we could parse and clean this data point. This shouldn't be hard, and would likely include the same or fewer challenges as cleaning addresses.

Documented queries:

Number of Unique Users:

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

Number of nodes and ways:

```
db.boston_massachusetts.aggregate([{"$group": {"_id": "$type", "count": {"$sum":
1}}}, {"$sort": {"count": -1}}])
```

Number of tag subtypes:

- Amenities:
  - `db.boston_massachusetts.aggregate([{"$group": {"_id": "$amenity", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}])`
- Shops:
  - `db.boston_massachusetts.aggregate([{"$group": {"_id": "$shop", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}])`
- Tourism:
  - `db.boston_massachusetts.aggregate([{"$group": {"_id": "$tourism", "count": {"$sum": 1}}}, {"$sort": {"count": -1}}])`

See `overview_stats.py` *for more.*

Number of node types for amenities, shops, and tourism locations:

Run `overview_stats.py` *for full details.*

- Top 5 Amenity Types:
  - 1314 - parking
  - 1034 - bench
  - 725 - school

    - 607 - restaurant
    - 446 - parking_space
- Top 5 Shop Types:
    - 96 - convenience
    - 76 - supermarket
    - 57 - hairdresser
    - 45 - alcohol
    - 43 - clothes
- Top 5 Tourism Types:
    - 96 - hotel
    - 53 - museum
    - 50 - artwork
    - 31 - viewpoint
    - 29 - attraction