1. Problems Encountered in the Map Data

There were several major issues present in the data as I initially cleaned it.

1) Missing street type:

For example, instead of entering “Baltimore Avenue”, the data might just say “Baltimore.” In order to correct this data, I used the auditing code created in the exercises in lesson 6 of the MongoDB course. This code takes the final word in the street name and creates an array of these with these words with the assumption that the final word in the street name is going to be a street type like the word “Street” or “Boulevard.” By looking at the output of that code, I was able to see what streets might potentially be missing their street types as they showed up as names like “Baltimore” or “Kasold” indicating that these were the final words in the street name. I then looked for these entries in the original XML file and cross referenced the other data from the node, like the name of the business, with Google Maps to determine the proper street type.

2) Inconsistent compass directions of street names:

Many of the street names included directions such as “North” in their names. These names were not written uniformly, so in the case of “North State Street,” on different occasions it might be written as “N State Street,” or “N. State Street.” In order to standardize these directions, I used regular expressions to find these strings in the xml data and programmatically standardized them as they were being converted into JSON so that I was left only with the full names of the directions and no abbreviations.

The following code was used automatically clean the data:

```
def update_direction(name, mapping):
        m = directions_re.search(name)
        if m.group(0) in mapping:
                name = re.sub(directions_re,mapping[m.group(0)],name)
        return name
```

Where “mapping” was a dictionary mapping variations of the directions found in the data into the standardized directions.

3) Mistaken or inconsistent store type naming:

When I initially wanted to look at store types, I ran this MongoDB query.

```
db.maps.aggregate ({"$group":{"_id":"$shop", "count":{"$sum":1}}},{"$sort":{"count":-1}})
```

This query aggregated and counted all shop types and then returned the results from most to least number of that shop type. Most of the data returned was for the number of shops of a given type. For example:

{u'_id': u'supermarket', u'count': 179}

{u'_id': u'books', u'count': 12}

I ordered the data from highest to lowest number because I assumed that typos and mistaken entries would appear with a count of 1 at the bottom of this list. Looking at the list, there were entries like:

{u'_id': u'US Toy', u'count': 1}
{u'_id': u"Hornbeck's Thriftway", u'count': 1}
{u'_id': u'Mission Farms Shopping Area', u'count': 1}

In these kinds of entries it was clear that instead of entering the store type, people were entering the actual store names. In these cases the data was amended to reflect the store's type rather than the store's name when possible.

More troubling in the data was the presence of the following results to the query:

{u'_id': u'yes', u'count': 698}

{u'_id': u'no', u'count': 1}

In this case, people were entering "Yes" to indicate that there was a store, but were not entering the type of store. Unfortunately there was no easy fix for this problem. Instead it was necessary to consider the implications of having this field have the store type in some cases and only indicating that it was a store in others. In particular, it is important to note that any count of a given store type is going to be less than or equal to the true number of stores of that type. For example, there were 153 stores with the value "Convenience" in the data. In addition to these 153 stores, we do not know how many of the fields marked "Yes" are also convenience stores. As a result, when reporting the data, it is necessary to say that there are at least 153 convenience stores. As a result of this analysis, I switched over to using information from the "Amenity" tag which simultaneously offers richer and more consistent data of a similar nature.

2. Data Overview

The map data utilized for this project was for the Kanses city region.
For many of the queries in this section the following code was used:

```
def aggregate(db, pipeline):
    result = db.maps.aggregate(pipeline)
    return result
```

Basic overview:

OSM File size:	404.483MB
JSON File size:	442.911MB

Unique Users:

Number of Unique Users: 782

```
def get_users_pipeline():
    pipeline = [{"$group":{"_id":{"user":"$created.user"}}}]
    return pipeline

users_pipeline = get_users_pipeline()
```

Nodes and Ways:

Number of Nodes:	1,636,090
Number of Ways:	205,540

```
def get_node_types_pipeline():
    pipeline = [{"$group":{"_id":"$node_type", "count":{"$sum":1}}}]
    return pipeline

result = aggregate(db, node_pipeline)
```

Amenities:

Number of Amenity Tags: 12,218

```
result = db.maps.find({"amenity":{"$exists":1}}).count()
```

Specific Amenities:

Number of Waste Baskets:	79
Number of Benches:	83
Number of Fountains:	52

```
def get_amenity_types_pipeline():
    pipeline = {"$group":{"_id":"$amenity", "count":{"$sum":1}}},{"$sort":{"count":-1}}
    return pipeline

amenity_pipeline = get_amenity_types_pipeline()

result = aggregate(db,amenity_pipeline)
```

Top 3 Most Common Amenities:

Parking:	2913
School:	1743
Place of Worship:	1446

```
def get_top_amenity_types_pipeline(number):
    pipeline = {"$match":{"amenity":{"$ne":None}}},{"$group":{"_id":"$amenity",
    "count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":number}
    return pipeline

top_amenity_pipeline = get_top_amenity_types_pipeline(3)

result = aggregate(db, top_amenity_pipeline)
```

3. **Potential Improvements to Data**

Looking at the overall data on amenities, it seems as though there are a few amenities that have extremely good coverage and many others that do not.
As an example out of the 12,218 nodes tagged as amenities the following types have extremely good coverage:

Parking spaces: 24% of total nodes (2913 marked)
Schools: 14% of total nodes (1743 marked)
Place of Worship: 12% of total nodes (1446 marked)

On the other hand, others have much worse coverage than one might expect:

Doctors: 0.3% of total nodes (35 marked)
ATM: 0.2% of total nodes (23 marked)

It seems that part of the problem might be that it's somewhat unclear what counts as an amenity and what the options for amenity are in general. It might be helpful data on the types of amenities that others have entered in list form in an easily accessible part of the website so that individuals interested in editing the maps have an idea of the types of things that they might be expected to enter. For example, it might not occur to a person to note down their local doctor's office until they see it on a list. This subtle change might increase the proportion of several amenities that currently appear to be underrepresented.

4. Other Fun Facts

Number of Distinct Streets:

I found the number of distinct streets using the following code:

```
def get_street_name_number_pipeline():
    pipeline =
    {"$match":{"address.street":{"$ne":None}}},{"$group":{"_id":"$address.street",
    "count":{"$sum":1}}},{"$sort":{"count":-1}}
```

```
    return pipeline


    main():
        street_name_number_pipeline = get_street_name_number_pipeline()
        result = aggregate(db, street_name_number_pipeline)
        total_streets = len(result["result"]
        print "Number of Unique Streets:", total_streets)
```

There were 2354 unique streets in the dataset. This code was run on data after the ordinal directions and street names had been standardized across the dataset to prevent double counting.

Most frequently mentioned streets:

I then decided to look at the top ten most mentioned streets in the dataset:

```
def get_street_names_pipeline(number):
        pipeline =
        {"$match":{"address.street":{"$ne":None}}},{"$group":{"_id":"$address.street",
        "count":{"$sum":1}}},{"$sort":{"count":-1}},{"$limit":number}
return pipeline
main():
        street_name_pipeline = get_street_names_pipeline(10)
        result = aggregate(db, street_name_pipeline)
        pprint.pprint(result)
```

Output:

```
{u'_id': u'Southwest Topeka Boulevard', u'count': 651},
{u'_id': u'Ohio Street', u'count': 573},
{u'_id': u'Tennessee Street', u'count': 570},
…
{u'_id': u'Louisiana Street', u'count': 422}
```

Number of streets only mentioned once:

I noticed that most of the streets in the dataset were only mentioned once. I decided to find out what percentage of them were only listed once with the following code:

```
def count_single_streets_pipeline():
        pipeline =
        {"$match":{"address.street":{"$ne":None}}},{"$group":{"_id":"$address.street",
        "count":{"$sum":1}}},{"$match":{"count":1}}
return pipeline
main():
        count_single_street_pipeline = count_single_streets_pipeline()
        result = aggregate(db, count_single_street_pipeline)
        single_streets = len(result["result"])
        print "Number of Streets Mentioned Once:", 100*single_streets/total_streets, "%"
```

Please note that the variable Total Streets was defined earlier

5. Conclusion

After looking at the data, it is clear that data on amenities is incomplete. The list of amenities I presented above makes it clear that a huge number of amenities are missing. As I presented earlier, there seemed to be an overrepresentation of parking lots and an underrepresentation of doctors' offices. This pattern indicates that the missing data is not missing in a random manner. In addition, while I did not conduct any analysis on the topic, it seems likely that the missing data is not spatially random either. It seems likely that some areas have better coverage of amenity data than others. As a result of this missing data, any type of statistical analysis we are likely to conduct will be extremely biased. Until the data is more complete, extreme caution should be exercised when drawing any sorts of conclusions from this data.