1. Data Wrangling Process

1. I first downloaded a osm file for SF Bay area.
2. As this file is big, it takes too long to test any code changes. I created a small sample file from the downloaded file
3. Counted all the tags from the downloaded file to see if it is different from the test file used during the class
4. Checked how many valid/invalid k value in each tag for the given data set.
5. Audited the street types by checking whether there are any unexpected street names or not. Whenever unexpected street names were found, I manually updated the mapping file to correct them.
6. Once everything works with the smaller sample size, repeated the process with the full map file.

2. Problems encountered

1. Street name audit: With larger data sets, there are more unexpected street types that need to be cleaned. I initially started creating mappings with a smaller sample (30MBytes) selected randomly from the original set, and I had to add a few more mappings of street names when I audited the entire file. I first found all the street types that were not expected, and created mappings by going through the unexpected street types manually.
2. City name audit: After importing the data to mongoDB, checked the number of city names in the data set. There are many incorrect city names due to typo, or unnecessary space at the end, starting with a lower case, state name being included. For example, San Francicsco, “berkeley”, “Artherton”, “Oakland, CA”, etc. Unlike street types, there are a lot of unique city names, and it requires more manual auditing and cleaning process.

Before cleaning up, there were 329 entries with “San Francicsco” as a city name.

```
db.sfmap.aggregate(
[
   {$match : {"address.city" : {"$exists" : true}}},
   {$project: {"city" : "$address.city"}},
   {$group: {"_id" : "$city" ,"count" : {"$sum":1}}},
   {$sort : {"count" : -1}}
]
)
      {
         "_id" : "San Francisco",
         "count" : 1041.0000000000000000
      },
      {
         "_id" : "San Francicsco",
         "count" : 329.0000000000000000
      },
```

```
{
    "_id" : "berkeley",
    "count" : 10.0000000000000000
},

{
    "_id" : "Artherton",
    "count" : 7.0000000000000000
},
```


To clean city names, I maintained expected city names, and mappings between correct and incorrect city names. I also removed the trailing spaces at the end and removed the characters after comma in the city name. Usually, the string after comma includes state or zipcode, but I didn't use them to set state or zip code, and I simply ignored them for this exercise.

3. Overview of the Data

1. Size of the file: 311MBytes.
2. Number of documents: 1564506
3. Number of unique users: 1288
4. Number of nodes with amenity: 13975
5. Number of unique type: 21
6. Top 5 users and amenities

Top 5 Amenities

Amenity	Number
parking	2958
restaurant	1857
school	1338
place_of_worship	1064
fire_hydrant	698

Top 5 users with most creations

User	Number
oldtopos	334076
KindredCoda	144613
osmmaker	140043
DanHomerick	119462
nmixter	77785

7. MongoDB queries

```
//Number of entries
db.sfmap.find().count()

//Number of unique users.
db.sfmap.distinct("created.user")

//Number of unique type
db.sfmap.distinct("type")

//Number of unique
db.sfmap.aggregate(
[
    {$match : {"created.user" : {"$exists" : true}}},
    {$group: {"_id" : "$created.user" ,"count" : {"$sum":1}}},
    {$sort : {"count" : -1}}
]
)
//Number of nodes with amenity
db.sfmap.find({"amenity" : {"$exists" : true}})

//Group by amenity
db.sfmap.aggregate(
[
    {$match : {"amenity" : {"$exists" : true}}},
    {$project: {"amenity" : "$amenity"}},
    {$group: {"_id" : "$amenity" ,"count" : {"$sum":1}}},
    {$sort : {"count" : -1}}
]
)
```

4. Other ideas about the datasets

Each entry includes location data (latitude, longitude), and this can be used to find locations with specific characteristics. For example, we could find a location with the largest number of restaurants within 1 mile.
Using location data will provide many interesting insights, but it may not be simple to do such analysis. Even though mongoDB provides a Geospatial index, it may not be straightforward to group and count the number of restaurants within 1 mile. We probably need to run “near” operator for every entry, and find the entry that has the most nearby entries. However, this method doesn't guarantee to give the maximum number because the center of 1 mile circle doesn’t have to be the location of existing entries.