Map Area: San Francisco, California

- Dataset downloaded from Map Zen
- I chose San Francisco because I work here, and I can relate to the data.

Program Code (Auditing and Cleaning Data):
https://github.com/Anumehra/Udacity/tree/master/Data%20Wrangling%20Final%20Project/Code

Problems encountered in your map

- Street types were abbreviated.
    - E.g. St., Ave, Blvd, etc.
    - Audited the street names and found out all the abbreviated street types.

```
mapping = { "St": "Street",
            "St.": "Street",
            "Rd.": "Road",
            "Rd": "Road",
            "Ave": "Avenue",
            "Ave.": "Avenue",
            "Ct": "Court",
            "Ln.": "Lane",
            "Blvd": "Boulevard",
            "Blvd.": "Boulevard",
            "Dr": "Drive",
            "Ctr": "Center",
            "Hwy": "Highway",
            "Dr.": "Drive",
            "Cres": "Crescent",
            "Plz": "Plaza” }
```

    - Used the function `update_street_type` to update the abbreviated street types.
- Postcode included alphabets, whitespaces, and special characters.
    - E.g. ['CA 94544', 'CA:94103', 'CA', 'CA94107', '94121 ']
    - Audited the postcode to found out the unexpected formats
    - Wrote the function `update_postcode` to remove the alphabets, whitespaces, and special characters.
- City names contained all numeric characters, and had inconsistent format.
    - E.g. [11720, 952, 157]
    - Wrote the function `update_city` to remove numerical values, and updated the city names to title case.
- Various abbreviated formats were used for state and country.
    - E.g. [California, ca, CA, california, US]
    - Made it consistent by assigning ‘CA’ for state and ‘US’ for country.

- Second level tags contain few key value pair with name “address”. This tag was conflicting with the address directory created within the node directory. It was reassigning the address directory to an object.
    - E.g. <tag k="address" v="400 Valencia Street"/>
    - Wrote a condition to ignore such tags.
- Second level tags contain few key value pair with name “postal_code”.
    - E.g. <tag k="postal_code" v="94105"/>
    - If addr:postcode does not exists but postcal_code exists, then assign postal code to postcode. For all other cases, ignore postal_code tag.
- I was assigning the type object to “Node” or “Way” in the beginning of the function `shape_element`, but it was getting reassigned to some other values like “Water”, ‘”Pump”, etc. I found out when I ran the query on Mongo DB to count the nodes and ways, by grouping ‘type’ and counting values. I was expecting just ‘nodes’ and ‘ways’. But, the result included couple of other types. While debugging, I figured out that there might be more than one attribute with name “type” at first level or second level tags. Therefore, I changed the code to assign the type object at the last so that it does not get reassigned.

Overview of the data

- Size of the file
    - san-francisco_california.osm: 658.2 MB
    - san-francisco_california.osm.json: 954.2 MB

- Number of unique users

    - Query:
      ```
      db.sanfran.aggregate([{$group : {_id : '$created.uid',
      count : {$sum: 1}}}, {$group : {_id : null, unique_users :
      {$sum: 1}}}])
      ```

      Result:
      ```
      { "_id" : null, "unique_users" : 2018 }
      ```

    - Alternate Query:
      Create Index: `db.sanfran.ensureIndex({"created.uid" : 1})`
      Get distinct: `db.runCommand({distinct: "sanfran", key: "created.uid"})`

Result:
```
"stats" : {
       "n" : 2014, /* No. of unique users */
       "nscanned" : 2014,
       "nscannedObjects" : 2014,
       "timems" : 53,
       "planSummary" : "DISTINCT { created.uid: 1.0 }"
  }
```

- Number of nodes and ways

Query:
```
db.sanfran.aggregate([{$group: {_id : '$type', count :
{$sum: 1}}}])
```

Result:
```
 { "_id" : "way", "count" : 329847 }
{ "_id" : "node", "count" : 3055692 }
```

- Number of chosen type of nodes, like cafes, shops, etc.

Query:
```
db.sanfran.aggregate([{$match : {'type' : 'node', 'amenity'
: {$ne : null}}}, {$group: {_id : '$amenity', count: {$sum:
1}}}, {$sort: {count : -1}}])
```

Result: (Sample result set for readability, not all values)
```
{ "_id" : "restaurant", "count" : 2089 }
{ "_id" : "place_of_worship", "count" : 744 }
{ "_id" : "cafe", "count" : 684 }
{ "_id" : "school", "count" : 654 }
{ "_id" : "post_box", "count" : 642 }
{ "_id" : "bench", "count" : 621 }
{ "_id" : "bicycle_parking", "count" : 446 }
{ "_id" : "fast_food", "count" : 410 }
{ "_id" : "drinking_water", "count" : 391 }
{ "_id" : "toilets", "count" : 332 }
{ "_id" : "bank", "count" : 288 }
{ "_id" : "parking", "count" : 277 }
{ "_id" : "fuel", "count" : 262 }
{ "_id" : "bar", "count" : 230 }
{ "_id" : "car_sharing", "count" : 226 }
{ "_id" : "pub", "count" : 182 }
{ "_id" : "post_office", "count" : 140 }
{ "_id" : "atm", "count" : 133 }
{ "_id" : "telephone", "count" : 120 }
{ "_id" : "pharmacy", "count" : 118 }
```

Other ideas about the datasets

- Get the list of top 10 cuisines in San Francisco area.

  Query:
  ```
  db.sanfran.aggregate([{$match : {'amenity' : { $in :
  ['restaurant', 'fast_food', 'cafe']}, 'cuisine' : {$ne :
  null}}}, {$group: {_id : '$cuisine', count : {$sum: 1}}},
  {$sort: {count : -1}}, {$limit: 10}])
  ```

  Result:
  ```
  { "_id" : "mexican", "count" : 235 }
  { "_id" : "coffee_shop", "count" : 206 }
  { "_id" : "burger", "count" : 184 }
  { "_id" : "pizza", "count" : 172 }
  { "_id" : "chinese", "count" : 130 }
  { "_id" : "sandwich", "count" : 122 }
  { "_id" : "italian", "count" : 117 }
  { "_id" : "japanese", "count" : 109 }
  { "_id" : "american", "count" : 104 }
  { "_id" : "thai", "count" : 91 }
  ```

- Find the postcode with the most number of apartments. This helps to narrow down the areas for apartment search.

  Query:
  ```
  db.sanfran.aggregate([{$match : {'building' : 'apartments',
  'address.postcode': {$ne : null}}}, {$group: {_id :
  '$address.postcode', count : {$sum: 1}}}, {$sort: {count :
  -1}}, {$limit: 10}])
  ```

  Result:
  ```
  { "_id" : "94610", "count" : 23 }
  { "_id" : "94109", "count" : 10 }
  { "_id" : "94107", "count" : 10 }
  { "_id" : "94606", "count" : 9 }
  { "_id" : "94404", "count" : 8 }
  { "_id" : "94015", "count" : 7 }
  { "_id" : "94063", "count" : 6 }
  { "_id" : "94102", "count" : 6 }
  { "_id" : "94501", "count" : 5 }
  { "_id" : "94103", "count" : 5 }
  ```

- Get the address of all the apartments in San Francisco city. Right now, I am looking for apartments in SF city, and this query provides the required data to start the search.

  Query:
  ```
  db.sanfran.find({'building' : 'apartments', 'address.city':
  'San Francisco'}, {address: 1})
  ```

  Result: (Sample result set for readability, not all values)
  ```
  {
    "_id" : ObjectId("55e2a9cbd55f096bb6da6298"),
    "address" : {
  ```

```
       "city" : "San Francisco",
       "street" : "Mission Street",
       "housenumber" : "1655",
       "postcode" : "94103"
  }
}
{
  "_id" : ObjectId("55e2a9cbd55f096bb6da68c6"),
  "address" : {
       "city" : "San Francisco",
       "street" : "Mission Rock Street",
       "housenumber" : "555",
       "postcode" : "94158"
  }
}
{
  "_id" : ObjectId("55e2a9ccd55f096bb6dab951"),
  "address" : {
       "city" : "San Francisco",
       "street" : "Market Street",
       "housenumber" : "1844",
       "postcode" : "94102"
  }
}
```

- For improving the dataset, we should try to extract data from one or more sources and validate it against each other. For example, if there exists a reliable data source with accurate street names (e.g. USGS map database), then we can normalize the open street database based on this reliable source. It will help us to build a more accurate dataset.