Data Analyst ND P3 | Data Wrangling with Mongo DB

Hai Xiao

Map Area: Santa Cruz, CA, United States

Site of prepared metro area from Map Zen:

https://mapzen.com/data/metro-extracts

Download link to Map Zen extracted Santa Cruz, CA OSM XML:

https://s3.amazonaws.com/metro-extracts.mapzen.com/santa-cruz_california.osm.bz2

1. Problems Encounted in the Map

After generating a smaller size sample data set, I have noticed following major issues, and some problem of my earlier submission in lesson 6, I will discuss altogether below:

- **First, code fix-up of lesson 6 submission.**
    - Generate arbitrary embedded dictionary other than "created" or "address".AS long as <tag k="%hash_name:%sub_key" v="%sub_val" /> is seen, an embedded dictionary %hash_name as "%hash_name": {"%sub_key": "%sub_val"} will be generated.

    - Code fix to generate arbitrary key/value pairs to <node> OR <way> element's hash. See "amenity", "tourism", "maxspeed", "operator", "bicycle", "highway" … from examples. (Original code strictly followed an explicit TODO lines in course problem set, so missed this out, and It's neither captured by the unit test case! Having this fixed, the TODO lines is updated to include new requirement explicitly - see submit p3.py line 330:333)

- **Second,** I have noticed a JSON dump problem wrt. UTF-8 chars from origibal data.py code. To be human readable and normalize to back end db process in case if fields are used, I have fixed the UTF-8 problem in Jason dump.
    - Character normalization - force UTF-8 encoding and JSON dump, so we will have:
      `"name": "Caffè Bene"  instead of "name": "Caff\u00e8 Bene"`
- **Third,** I have further noticed inconsistent ways of street info presentation in the chosen Santa Cruz area, particularly where Tiger Map Data import is used - in most cases when
  ```
      <tag k="tiger:..." /> exists in <way>
  ```
  the element would not have
  ```
      <tag k="addr:street" .. />,
  ```
  even though correct address:street can be extracted from the embedded hash tiger:{} generated.

  So I I have these work introduced here:

  Consolidate different ways to present street info in elements of type <way> in original JSON:

    - Extracting street info from 2nd level hash - 'tiger' whenever possible and needed.

Notice that, it is absolutely ok to skip this consolidation on JSON giving to MongoDB for backend data analysis, but it must be better to provide more uniformed JSON schema (of <way>) data to mongoimport, even I can use mongodb update $set cmd to achieve certain level data uniformness later, I believe do it beforehand in JSON is better, once data clean is done earlier in data pipeline, we can use mongodb focusing on analysis process in discover intelligence.

Scope of this improvement:

Only insert key:"street" value pair in constructed "address" hash for now. I could also extract more info (e.g. postcode) from tiger hash to "address", but that can be left for the future enhancement.

- **Fouth, fix inconsistent postal codes.**
  From XML to JSOM parsing (in a working in progress p3.py with 1.~3. above done), generated address hash may contain following inconsistent postcode strings:

  ```
          "95060", "95066-5121", "CA 95062", "CA95061"
  ```

  To benefit later data process I decide to keep the L most 5-digit and strip off remain.

- **Fifth, fix up inconsistent maxspeed presentation.**
  From OSM XML, most type:way elements had correct second level element for maxspeed, e.g:

  ```
      <tag k="maxspeed" v="65 mph" />
  ```

  But there are some not getting very correct, e.g:

  ```
      <tag k="maxspeed" v="25" />
  ```

  Our goal is to check and add proper speed unit to the maxspeed if it is missing.

  Heuristically I use 'mph' as unit since that's the most common case cross the document.

  I normalize the generated JSON (in shape_element) vs. modifying the XML input file.

- **Sixth, fix up inconsistent Bank and ATM data, and consolidate them.**

  6.1 to ATMs around the area, there are two different data presentations:

  ```
  Type I is: {
      "amenity": "bank", "atm": "yes", ... ...
  }
  Another Type II is: {
      "amenity": "atm",  ... ...
  }
  ```

  So it makes sense to unify these, I unified these to Type I to keep an explicit 'bank' relation.

  6.2 Another data inconsistent in the bank (or atm) data entries of element is that some have both "name" and "operator", some have "name" only, and others have only "operator". Making this consistent only helps later process whenever needed, so I will also add a "name" key value pair with the value from "operator" key whenever "name" is not present.

- **Seventh, integrate a street name Audit and Auto Update on consolidated address:street.**

  I have extended audit from course work to include:

  - Handle Street prefix or suffix, such as 'East', 'South', 'West', 'North', 'Extension', etc.
  - Handle Non-standard street type in both front & tail, e.g. 'Hwy 9', 'Caborillo Hwy.', etc.
  - Handle the mix of above two, such as 'Ocean St Extension', etc.
  - Auto-fix an abbr. street type to norm. one, e.g. 'Ocean St West' => 'Ocean Street West'
  - Above all, build a cross elements street=>street_type map (dict.) for reference when street_type isn't available within an element but clearly resolvable from other instances!

2. Data Overview

This section contains basic statistics about the dataset and the MongoDB queries used to gather them.

File sizes (Bytes)

```
santa-cruz_california.osm ......... 54,800,278
santa-cruz_california.osm.json .... 78,685,625
sample.osm ........................ 5,526,074
sample.osm.json ................... 7,852,206
```

MongoDB Import from JSON and db collection names

```
mongoimport --db osm --collection santa --file santa-cruz_california.osm.json
```

Main statistics and analysis

- Number of documents (all elements)

```
> db.santa.find().count()
 278695
```

- Number of nodes

```
> db.santa.find({"type":"node"}).count()
 257661
```

- Number of ways

```
> db.santa.find({"type":"way"}).count()
 20954
```

- Extra elements unknown?

```
 Number of documents - Number of nodes - Number of ways = 80
> db.santa.find({'type': {$nin: ['node', 'way']}}).count()
 80
```

- Other elements of other types

```
> db.santa.distinct("type")
 [
         "node",
         "conifer",
         "electronics",
         "cardboard",
         "drinks",
         "way",
         "water",
         "bocce",
         "Private",
         "lumber_yard",
         "H",
         "LB",
         "U",
         "F",
         "S-E",
         "C",
         "R",
                "PO",
                "O",
                "S-J",
                "wastewater",
                "multipolygon"
        ]
        > db.santa.distinct("type").length
        22
```

- Why do we have other types of elements and what are those?
  - Code only emits JSON obj from XML element of either 'node' or 'way', while iterparse
  - for XML element taged 'node', I created a JSON element with 'type': 'node' (as required)
  - for XML element taged 'way', I created a JSON element with 'type': 'way' (as required)
  - for an XML element ('node' or 'way'), we insert a property k_val:v_val to generated JSON object from XML element's arbitrary second level:

    ```
        <tag k=k_val v=v_val>
    ```
  - So why we could end up with more than TWO type values in JSON objects EXPLAINs: It is because some XML elements ('node'/'way') already has an second level sub-element has k='type' as

    ```
              <tag k="type" v="S-J"/>
                   in
       <way>
                ... ... ...
              <tag k="ID" v="119"/>
              <tag k="name" v="Shoreline Middle School"/>
              <tag k="type" v="S-J"/>
              <tag k="amenity" v="school"/>
              <tag k="Shape_len" v="2893.25879249"/>
              <tag k="Shape_area" v="533668.457339"/>
              <tag k="description" v="JUNIOR HIGH SCHOOL"/>
       </way>
    ```
    This internal {‘type’: ‘S-J’} is overwritten an outer {‘type’: ‘way’} or {‘type’: ‘node’}!
  - It is not merely an implementation Bug, since by design requirement we have not planned this type of data representation collision in JSON (or any other structured data reorganization) due to arbitrary data information.
  - To fix this issue, a data re-organization plan is anticipated.
  - But lessons are well learned! Whenever working on a data wrangling, we need to plan for potential data collision when we are introducing, modifying or relocating properties!
  - We now know that there exist 80 elements come with special

    ```
              <tag k='type' v=one_of_special_type_values />
    ```
- Cities within the map area

```
       > db.santa.distinct("address.city")
        [
            "Santa Cruz",
            "Scotts Valley",
            "Felton",
            "Aptos",
            "Capitola",
            "Soquel",
            "CAPITOLA",
            "SANTA CRUZ",
            "Bonny Doon"
        ]
        > db.santa.distinct("address.city").length
        9
```

- We could see that there are 7 cities (2 are duplicates above, and I can do further data de-duplication) in the map, so it is not just for Santa Cruz City, but Santa Cruz County!

3. Additional Ideas

This section contains additional data wrangling suggestion from previous process and analysis, normally data wrangling and analysis is a closely looped process, as we just see, it is very unlikely one round of data preparation will serve all the analysis, sometimes analysis process will expose that more data re-org or cleaning are necessary. This section will also include some extra idea wrt. using existing data set for further data discovery and validation, last but not least, some additional data exploration with MongoDB is included.

Additional Data Exploration and Problem Resolving

- **Highest and Lowest place in the area**
  - Using 'ele' (elevation) property from feasible documents, most likely we could find out highest and lowest places in the area. I won't say this result is authorative, as it is purly depend on data entries completeness and correctness, and we already know that not every entry has 'ele' provided.

    (Use data available only) A dry-run found a place above 600 ft:

```
> db.santa.find("this.ele > 600").limit(1)
 { "_id" : ObjectId("5688761c28538f1f91bd55a7"), "created" : { "changeset" :
"8178925", "user" : "nmixter", "version" : "1", "uid" : "55774", "timestamp" :
"2011-05-18T08:37:02Z" }, "type" : "node", "pos" : [ 37.0613703, -122.0317029 ],
"ele" : "640", "tourism" : "viewpoint", "id" : "1290029877" }
```

    Then to find the highest place from data set:

```
> db.santa.aggregate([{$match: {"ele": {$exists: 1}}}, {$sort: {"ele": -1}},
{$limit: 1}])
 { "_id" : ObjectId("5688761a28538f1f91bb3141"), "amenity" : "school", "name" :
"Westlake Elementary School", "created" : { "changeset" : "1309292", "user" :
"fennecfoxen", "version" : "3", "uid" : "100744", "timestamp" : "2009-05-
24T19:05:06Z" }, "pos" : [ 36.97857, -122.0490403 ], "ele" : "99", "type" :
"node", "id" : "358849229", "gnis" : { "feature_id" : "1872863", "state_id" :
"06", "county_id" : "087", "created" : "06/13/2000" } }
```

  - So WHAT IS THE PROBLEM?

    Why the sound db collection aggregate only returns "99" as highest "ele", while there is place above "640"?

    It is quiet obvious that a string vs. a number comparison was taken place!

    It is because the data type of ele's value is stored as STRING in DB, inherented from imported JSON.

    MongoDB MongoDB can't sort by numbers stored as strings. I either have to store them as number starting in JSON before DB import or convert (migrate) the field to number in MongoDB afterwards. Then sort would work as expected!

    Choose the 2nd approach in this excercise:

    (convert 'ele' from string to float)

```
> db.santa.find({'ele': {$exists: 1}}).forEach(function(doc) {
...    doc.ele = parseFloat(doc.ele);
...    db.santa.save(doc);  ... });
>
```

Now check out the highest place :)

```
> db.santa.aggregate([{$match: {"ele": {$exists: 1}}}, {$sort: {"ele": -1}}, {$limit: 2}])
  { "_id" : ObjectId("5688761a28538f1f91bb5c33"), "name" : "Observation Deck", "created" : { "
changeset" : "3411231", "user" : "DanHomerick", "version" : "3", "uid" : "160138", "timestamp"
 : "2009-12-20T03:25:48Z" }, "tourism" : "viewpoint", "pos" : [ 37.026178, -122.046024 ], "ele
" : 800, "type" : "node", "id" : "411642604" }
  { "_id" : ObjectId("5688761c28538f1f91bd55a7"), "created" : { "changeset" : "8178925", "user
" : "nmixter", "version" : "1", "uid" : "55774", "timestamp" : "2011-05-18T08:37:02Z" }, "type
" : "node", "pos" : [ 37.0613703, -122.0317029 ], "ele" : 640, "tourism" : "viewpoint", "id" :
 "1290029877" }
```

    - Now this result sounds correct: 800 and 640 are the top 2 in elevation, both are viewpoints, tourism related! - Those are most likely over the Santa Cruz Mountain.
- **Lowest** places in the area.
  (Knowing Santa Cruz is a coastal area, lowest places must be beach, bay or water cove)

```
 > db.santa.aggregate([{$match: {"ele": {$exists: 1}}}, {$sort: {"ele": 1}}, {$limit: 10}])
  { "_id" : ObjectId("5688761a28538f1f91bb338a"), "natural" : "bay", "name" : "Soquel Cove",
"created" : { "changeset" : "150679", "user" : "amillar", "version" : "2", "uid" : "28145",
"timestamp" : "2009-04-03T22:48:45Z" }, "pos" : [ 36.9657842, -121.9307918 ], "ele" : -3,
"address" : { "state" : "CA" }, "gnis" : { "feature_id" : "253911", "county_name" : "Santa
Cruz", "feature_type" : "Bay", "created" : "01/19/1981" }, "type" : "node", "id" :
"369165862", "source_ref" : "geonames.usgs.gov" }
    ... ... ...
  { "_id" : ObjectId("5688761a28538f1f91bb3390"), "natural" : "bay", "name" : "Sunny Cove",
"created" : { "changeset" : "4966208", "user" : "ZekeFarwell", "version" : "5", "uid" :
"38090", "timestamp" : "2010-06-11T21:19:50Z" }, "pos" : [ 36.959989, -121.9895577 ], "ele" :
1, "address" : { "state" : "CA" }, "gnis" : { "feature_id" : "1723315", "county_name" : "Santa
Cruz", "feature_type" : "Bay", "created" : "03/05/1997" }, "type" : "node", "id" :
"369172617", "source_ref" : "geonames.usgs.gov" }
  { "_id" : ObjectId("5688761b28538f1f91bb8e5f"), "name" : "KSCO-AM (Santa Cruz)", "created"
: { "changeset" : "1778239", "user" : "stevea", "version" : "3", "uid" : "123633", "timestamp"
: "2009-07-09T07:35:44Z" }, "man_made" : "tower", "pos" : [ 36.9619737, -121.9812269 ], "ele"
: 1, "type" : "node", "id" : "437890177", "gnis" : { "feature_id" : "1662748", "state_id" :
"06", "county_id" : "087", "created" : "07/01/1994" } }
  { "_id" : ObjectId("5688761a28538f1f91bb3107"), "name" : "KSCO-AM (Santa Cruz)", "created"
: { "changeset" : "1778239", "user" : "stevea", "version" : "6", "uid" : "123633", "timestamp"
: "2009-07-09T07:35:44Z" }, "man_made" : "tower", "pos" : [ 36.9617405, -121.9816389 ], "ele"
: 1, "type" : "node", "id" : "358810162", "gnis" : { "feature_id" : "1662748", "state_id" :
"06", "county_id" : "087", "created" : "07/01/1994" } }
    ... ... ...
  { "_id" : ObjectId("5688761b28538f1f91bb8e63"), "name" : "KSCO-AM (Santa Cruz)", "created"
: { "changeset" : "1778239", "user" : "stevea", "version" : "3", "uid" : "123633", "timestamp"
: "2009-07-09T07:35:44Z" }, "man_made" : "tower", "pos" : [ 36.9615554, -121.9820251 ], "ele"
: 1, "type" : "node", "id" : "437890173", "gnis" : { "feature_id" : "1662748", "state_id" :
"06", "county_id" : "087", "created" : "07/01/1994" } }
  { "_id" : ObjectId("5688761a28538f1f91bb30e2"), "name" : "Capitola State Beach", "created"
: { "changeset" : "781903", "user" : "iandees", "version" : "1", "uid" : "4732", "timestamp" :
"2009-03-11T05:26:53Z" }, "pos" : [ 36.971895, -121.9494038 ], "leisure" : "park", "ele" : 2,
"type" : "node", "id" : "358763055", "gnis" : { "feature_id" : "220587", "state_id" : "06",
"county_id" : "087", "created" : "01/19/1981" } }
```

    - This confirmed my guess, but also disclosed a problem - i.e. data duplication:
        - Check out 3 documents all with:

          ```
                {"name" : "KSCO-AM (Santa Cruz)", "uid" : "123633"}
          ```

          They are created by the same "user":"stevea" on the same day, but with different gps coordinates and versions.
        - So what should we trust?
          This is a very representative problem in bigger scope of many data set!
        - **Suggestion to improvement** (but not to code up here) in this cases:

          Todo data de-duplication in the business logic

            - Find duplication of multiple occurance, 'uid' in this cases
            - Decide which unique document to pick for business logic, latest 'version', e.g.
            - Only used the picked (versioned) ones in active logic (query, aggregate, etc.)

- **Banks and ATMs exploration in the area**
    - Previously I have tried to normalize BANK and ATM data in JSON, so every document would possibly have a 'name' of banking company vs. some have them only kept in 'operator' field. After import that improved JSON, the work below is more straightforward.
- Most presented BANKs in the area

```
> db.santa.aggregate([
...    {$match: {'amenity': {$exists: 1}, 'amenity': 'bank'}},
...    {$group: {'_id': '$name', 'count': {$sum: 1}}},
...    {$sort: {'count': -1}}
... ])
{ "_id" : null, "count" : 25 }
{ "_id" : "Bank of America", "count" : 12 }
{ "_id" : "Bay Federal Credit Union", "count" : 11 }
{ "_id" : "Wells Fargo", "count" : 6 }
{ "_id" : "Chase", "count" : 6 }
{ "_id" : "Comerica", "count" : 2 }
{ "_id" : "Santa Cruz County Bank", "count" : 2 }
{ "_id" : "Comerica Bank", "count" : 2 }
{ "_id" : "US Bank", "count" : 2 }
{ "_id" : "Santa Cruz Community Credit Union", "count" : 1 }
{ "_id" : "Liberty Bank", "count" : 1 }
{ "_id" : "Lighthouse Bank", "count" : 1 }
{ "_id" : "Bank Of America", "count" : 1 }
{ "_id" : "CashPlus", "count" : 1 }
{ "_id" : "Union Bank", "count" : 1 }
{ "_id" : "Citibank", "count" : 1 }
{ "_id" : "Monterey Credit Union", "count" : 1 }
{ "_id" : "Rabobank", "count" : 1 }
{ "_id" : "Bank of the West", "count" : 1 }
{ "_id" : "Wells Fargo Bank", "count" : 1 }
{ "_id" : "Mountain Roasting Company", "count" : 1 }
{ "_id" : "Western Union", "count" : 1 }
{ "_id" : "Wachovia", "count" : 1 }
>
> db.santa.find({'amenity': {$exists: 1}, 'amenity': 'bank'}).count()
82
```

    - Here we found list of BANKs with presenting frequence in descending order. A problem seen - "Bank of America" and "Bank Of America" were thought different! This is the same problem as we see in the query of "Cities within the map area". This tells us again: it is very necessary to normalize String presentation prior to analysis
    - From result, there are 25 banks without a name, I need to find out what are those: (following queries tell those 25 are ATM only sites vs. Bank offices, and it explains :)

```
> db.santa.find({'amenity': {$exists: 1}, 'amenity': 'bank', 'name': {$exists: 1} }).count()
57
>
> db.santa.find({'amenity': {$exists: 1}, 'amenity': 'bank', 'name': {$exists: 0} }).count()
25
>
> db.santa.find({'amenity': {$exists: 1}, 'amenity': 'bank', 'name': {$exists: 0} }).limit(2)
  { "_id" : ObjectId("5688761b28538f1f91bb7785"), "amenity" : "bank", "created" : { "changeset"
: "1477131", "user" : "stevea", "version" : "1", "uid" : "123633", "timestamp" : "2009-06-
10T12:58:34Z" }, "atm" : "yes", "pos" : [ 36.9603116, -122.0203379 ], "type" : "node", "id" :
"418204220" }
  { "_id" : ObjectId("5688761b28538f1f91bb8d4e"), "amenity" : "bank", "created" : { "changeset"
: "25989257", "user" : "njaard", "version" : "2", "uid" : "173918", "timestamp" : "2014-10-
10T18:59:18Z" }, "pos" : [ 36.9811039, -122.0058258 ], "address" : { "street" : "Soquel Avenue",
"housenumber" : "1414" }, "type" : "node", "id" : "436631688" }
>
> db.santa.find({'amenity': {$exists: 1}, 'atm': 'yes' }).count()
58
>
> db.santa.find({'amenity': {$exists: 1}, 'atm': 'yes', 'amenity': 'bank' }).count()
58
```

    - This also tells that there are 58 ATM locations in total.

Extra Thought on improving the data from previous process and beyond

- Normalize String Presentation (in addition to UTF-8 handling) for many cases
  - We have seen that "CAPITOLA" and "Capitola", "Santa Cruz" and "SANTA CRUZ", "Bank of America" and "Bank Of America" are all unnecessarily treated as distinct value if further normalization is NOT done!
    It produces duplicated query results, so it makes sense to clean this in data wrangling!
  - Feasible solution can be as simple as use .lower() or .upper() method on the strings in question (on JSON update prior to mongodb import), to a little more elegant ways.
- Try to uniform data presentation as much as possible
  - In this and may other data sets, you will most likely see a similar property (of JSON) is presented in different data hierarchy (level) from different documents. Extract, re-org and put them up in an uniformed access hierarchy is very import to the later on data query and pipeline process. This requires extensive and very careful data mangling work beforehand, but convenience at end will pay off.
  - In the code submission, I have done similar work to:
    - Formalized 'address':'street' presentation from 'tiger' data
    - Formalized on 'name' attribute from 'operator' in bank documents
- Pay attention to data type (field) and supported mongodb operations
  - Always choose the most suitable data type importing to DB. It is much cheaper to convert string to number in JSON than a DB migration, e.g. This work is exposed in the 'elevation' exercise above.
- Very importantly always plan for the proving data consolidation JSON schema at processing arbitrary data inputs, be extremely aware of potential data collision or overwrite from the presentation of you choice, and plan for what todo when it happens.
  - We have seen this iisue of the property 'type' of each document in the beginning.
  - Do not let this problem found later. Worthywhile try to address in the very start.
- Data de-duplication, many data set has multiple but versioned entries of the same document (object), some are older ones, some are inaccurate ones, use them all without filter in any business logic is an error. We should find ways to use ONLY proper one at a given time! and validation, such as the Plan for the 'type' when come up data/JSON schema, be aware of collision Normalize String cases in various locations
  - We have seen this in the 'elevation' exercise that 3 dups in lowest 10 elevations.
  - And this type of dups exist in many other places or data sets as well.
  - To be flexible, fix is most likely in the DB query functions vs. purge in JSON.
- Last but not least, Data validation, Audit and Update.
  - Data can be dirty and inconsistent, contain human errors from the beginning.
  - Have a trustable data to use is vital important to any business logic!
  - Remove/minimize data error or inconsistency is an important part of data wrangling.
  - Audit is an efficient method to discovery data distribution or problems.
  - Valid data further requires understanding of document internal and cross document context, some times even requires understanding multiple data sets or data sources.

Implementing Data Improvement, Anticipated Problems and Discussion

- **Take 'Normalize String Presentation' above as an example to implement data improvement.**
  I could implement in two different ways (below) for an improved result.
  - Convert all bank name strings to either upper or lower case in db (direct update) This is similar to what I have done in the 'elevation' exercise earlier, there I converted 'ele' value from string to number and updated db record (document) one by one. This should work as expected, but there would be two problems anticipated:
    - Performance issue: as the db update is per document vs. in group, that implementation is NOT performance optimized, in case of big database it will not be preferred.
    - Overwrite of original data, while it is an improvement necessary for an analysis, we might not want to overwrite original data source too often, to keep non-error data as it is has some value in certain circumstances.

   - ***No db update with string Conversion, instead use "$project" to normalize bank names in the middle stage of pipeline*.**
     "$project" is an important feature of mongodb that can be used to prepare data as needed without modifing original data set in db. Since it addresses both concerns from method one, I will go ahead give a try:

- **Most presented BANKs in the area (Using $project)**

```
> db.santa.aggregate([
 ...   {$match: {'amenity': {$exists: 1}, 'amenity': 'bank'}},
 ...   {$project: { uname: {$toUpper: '$name'} }},
 ...   {$group: {'_id': '$uname', 'count': {$sum: 1}}},
 ...   {$sort: {'count': -1}}
 ... ])
 { "_id" : "", "count" : 25 }
 { "_id" : "BANK OF AMERICA", "count" : 13 }
 { "_id" : "BAY FEDERAL CREDIT UNION", "count" : 11 }
 { "_id" : "WELLS FARGO", "count" : 6 }
 { "_id" : "CHASE", "count" : 6 }
 { "_id" : "COMERICA", "count" : 2 }
 { "_id" : "COMERICA BANK", "count" : 2 }
   ......... .........
 { "_id" : "SANTA CRUZ COMMUNITY CREDIT UNION", "count" : 1 }
 { "_id" : "WACHOVIA", "count" : 1 }
 { "_id" : "LIBERTY BANK", "count" : 1 }
 { "_id" : "RABOBANK", "count" : 1 }
 { "_id" : "WESTERN UNION", "count" : 1 }
 { "_id" : "BANK OF THE WEST", "count" : 1 }
 { "_id" : "UNION BANK", "count" : 1 }
 { "_id" : "CITIBANK", "count" : 1 }
 { "_id" : "MONTEREY CREDIT UNION", "count" : 1 }
 >
```

   - As anticipated:
     - No more different groups for 'Bank of America' and 'Bank Of America'. So this is a better.
       This method/techinique has a broader use to address many use cases, e.g.
       It can be used to de-duplicate result of distinct cities found!
     - Only dups at 2nd order degree exist, such as 'COMERICA' and 'COMERICA BANK'. Further We can implement similar technique from course work (audit.py) to recognize some common prefix or suffix in names during data process.

4. Conclusion

I have only done some of the initial work to make the data very informational, trustable and convenient to business logic. But I already learned an experience that data wrangling takes lots of time to make data right and better, and it may require keep doing for several rounds.
I'm also strived to find methods to valid data entries as much as possible, but only a smaller portion is possibly captured in this submission. There may be more natural or innovative ways to verify data in data set, once I learned more or gained more experience in OSM.

Currently, I'm interested in following idea:

```
    - Use the association between 'node' and 'way' for data mining
    - Bring in 'relation' to the data mining
    - Try to visualize a map from data set
    - Try to apply DB updates from periodic data source (OSM) pulling.
    - Try to validate data element cross site (or source), e.g. use google
    map api to confirm gps coordinates, cross check street names, address
    numbers, etc.
```

I believe a practical application can be built, once most of these interests are well addressed.