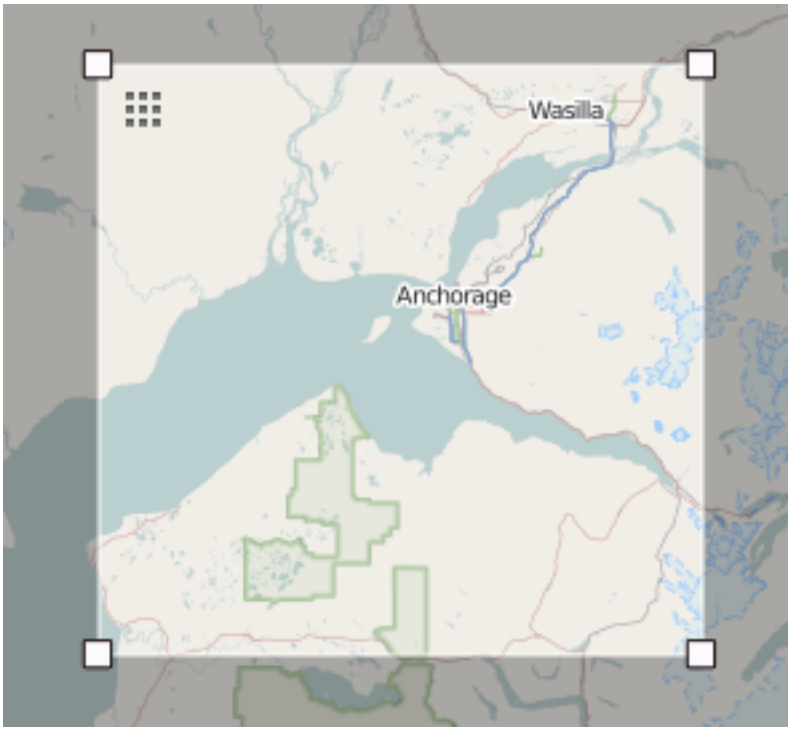
OpenStreet Map Project Data Wrangling with MongoDb

Project Summary
Craig Nicholson
craig.nicholson@gmail.com

Table of Contents

Map Area

Anchorage, Alaska, and surrounding areas, United States. Anchorage is the largest city in Alaska in regards to population.



Map bounds:

```
<bounds
      minlat="58.757"
      minlon="-152.831"
      maxlat="62.047"
      maxlon="-145.92"/>
```

URL: http://www.openstreetmap.org/export#map=8/61.107/-150.117

The Overpass API to download the openstreet map data file.

Problems Encountered in the Map

A 124 MB data file was downloaded using the Alaska area. The main issues with this data are the following:

- Inconsistent Post Codes.
- Abbreviated Street Names.
- Incorrectly spelled City names.
- Multiple 'created' keys { gnis:created } within the data.

Inconsistent Post Codes

The postcodes are formatted to only have numerical values and to be empty, 5, or 9 character integers. The typically US postal code consists of a 5 digit number, and in some cases is also represented as a 5 digit number separated by a dash (-) and preceded by a 4 digit number.

```
def update_zipcode(zipcode):
    '''
        Clean the zip code
        These are a few of the errors one might encounter
        { "_id" : "Homer, AK 99603", "count" : 2 }
        { "_id" : "AK", "count" : 1 }
        { "_id" : "Alaska", "count" : 1 }
        { "_id" : "AK 99501-2129", "count" : 1 }
        { "_id" : "AK 99501-2118", "count" : 1 }
    '''
    # use regex to remove all strings from zipcode, this
    # will leave us with a numeric number which should be
    # 5 or 9 characters long
    zipcode_clean = re.sub(r"\D", "", zipcode)

    return zipcode_clean
```

The postal codes are cleaned using a regex to remove any strings leaving the integers. More error checking and cleaning could be done to make sure each postcode value is a 5 or 9 digit number.

```
db.alaska.aggregate(
        {
                $match:{ "address.postcode" : {$exists:1}}
        },
        {
                $group:{_id:"$address.postcode","count":{$sum:1}}
        },
        {
                $sort:{count:-1}
        }
)
{ "_id" : "99501", "count" : 19 }
{ "_id" : "99515", "count" : 17 }
{ "_id" : "99503", "count" : 15 }
{ "_id" : "99669", "count" : 14 }
{ "_id" : "99603", "count" : 13 }
{ "_id" : "99572", "count" : 10 }
{ "_id" : "99577", "count" : 7 }
{ "_id" : "99518", "count" : 5 }
{ "_id" : "99507", "count" : 5 }
{ "_id" : "99664", "count" : 5 }
{ "_id" : "99504", "count" : 3 }
{ "_id" : "99508", "count" : 3 }
{ "_id" : "99516", "count" : 3 }
{ "_id" : "99517", "count" : 3 }
{ "_id" : "99654", "count" : 3 }
{ "_id" : "", "count" : 2 }
{ "_id" : "99645", "count" : 2 }
{ "_id" : "99611", "count" : 2 }
{ "_id" : "99587", "count" : 1 }
{ "_id" : "99610", "count" : 1 }
{ "_id" : "99693", "count" : 1 }
{ "_id" : "995012118", "count" : 1 }
{ "_id" : "99502", "count" : 1 }
{ "_id" : "995012129", "count" : 1 }
```

Abbreviated Street Names

Several abbreviated street names where found and corrected using python. Below are a few examples from the data set.

Street Name Original	Street Name Corrected / Cleaned
West Evergreen Ave.	West Evergreen Avenue
Ashwood	Ashwood Street
Merrill Field Dr	Merrill Field Drive
East Northern Lights Blvd.	East Northern Lights Boulevard

Incorrectly Spelled City Names

The address block had cities with a misspelled city name.

City Name Original	City Name Corrected / Cleaned
Anchoage	Anchorage

Multiple 'created' keys { gnis:created } within the data.

The data.py code would find similar keys to the "addr:" key and process these keys like generic keys.

I had to review the gnis:created key, and any keys whose text to the right of the colon was 'created,' because the parser would remove the the text to the left of the colon and keep the text to the right of the colon. The gnis:created key value pair came after the created object had been populated and the creating 'created' from the 'gnis:created' which would overwrite the first created tag.

Note the node's timestamp is 2009 and the gnis:created datea is 2000, so I am assuming the gnis data is old or the date in the value was a default date.

The oldest timestamp in this dataset dates back to 2007, which indicates a inconsistency in this tag's value.

```
<node id="369141794" lat="61.0058333" lon="-147.0102778" version="1"
timestamp="2009-04-03T04:23:53Z" changeset="112872" uid="28145"
user="amillar">
   <tag k="addr:state" v="AK"/>
   <tag k="ele" v="24"/>
   <tag k="gnis:county_name" v="Valdez-Cordova (CA)"/>
   <tag k="gnis:created" v="01/01/2000"/>
   <tag k="gnis:feature_id" v="1403286"/>
   <tag k="gnis:feature_type" v="Bay"/>
   <tag k="name" v="Heather Bay"/>
   <tag k="natural" v="bay"/>
 </node>

 <tag k="gnis:created" v="01/01/2000"/>
```

Additional notes when reviewing the data

Alternate ways of cleaning the data:

- Notify the users who entered the data so they can fix the source of the problem.
- Capture the nodeID and update the addresses using the OpenStreetMap API to push and update.
- After you have pushed the change and the change is accepted you can re-pull the data.

Pushing changes to OpenStreetmap using the API

- http://wiki.openstreetmap.org/wiki/API_v0.6#Update:_PUT_.2Fapi.2F0.6.2F.5Bnode.7Cway.7Crelation.5D.2F.23id
  - Update: PUT /api/0.6/[node|way|relation]/#id
  - Delete: DELETE /api/0.6/[node|way|relation]/#id
- OpenStreetmap handles changesets so using the API would be a good solutions. http://wiki.openstreetmap.org/wiki/API_v0.6#Changesets_2

Problem Character Encountered

- <tag k="theregister.co.uk" v="black helicopters"/>

Dropping of the 'key' on keys which have a colon

The algorithm assumes any key with a colon we can just drop the text to the left of the colon. For the Alaska dataset we this we lose some fidelity. In this example we can see the <tag k="FIXME:bicycle" v="verify"/> which lets us know this tag needs work.

```
 <way id="8903862" version="5" timestamp="2012-10-04T03:08:25Z"
changeset="13355593" uid="207745" user="NE2">
  <nd ref="566515911"/>
  <nd ref="1287563183"/>
  <nd ref="1287563474"/>
  <nd ref="1287563365"/>
  <nd ref="566515912"/>
  <nd ref="1287563233"/>
  <nd ref="566527136"/>
  <nd ref="65165950"/>
  <tag k="FIXME:bicycle" v="verify"/>
  <tag k="bicycle" v="yes"/>
  <tag k="highway" v="motorway_link"/>
  <tag k="oneway" v="yes"/>
  <tag k="tiger:cfcc" v="A63"/>
  <tag k="tiger:county" v="Anchorage, AK"/>
 </way>
```

A total of 65 'FixMe' tags where found.

Example of a tag where the Fixme was removed.
<tag k="FIXME:bicycle" v="verify"/>

```
db.alaska.find({id:'8903862'}).pretty()
{
        "_id" : ObjectId("54e96327a0f587a3a9d1bec3"),
        "bicycle" : "yes",
        "created" : {
                "changeset" : "13355593",
                "user" : "NE2",
                "version" : "5",
                "uid" : "207745",
                "timestamp" : "2012-10-04T03:08:25Z"
        },
        "pos" : [
                0,
                0
        ],
        "cfcc" : "A63",
        "county" : "Anchorage, AK",
        "oneway" : "yes",
        "node_refs" : [
                "566515911",
                "1287563183",
                "1287563474",
                "1287563365",
                "566515912",
                "1287563233",
                "566527136",
                "65165950"
        ],
        "type" : "way",
        "id" : "8903862",
        "highway" : "motorway_link"
}
```

Data Overview

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

Table 1. File sizes

Alaska.xml	124.2 MB
Alaska.xml.json	138.6 MB

Number of documents

```
db.alaska.find().count()
```

622579

```
db.alaska.stats()
{
      "ns" : "osm.alaska",          // namespace
      "count" : 622579,             // number of documents
      "size" : 179039952,           // collection size in bytes
      "avgObjSize" : 287,           // average object size in bytes
      "storageSize" : 243314688,    // (pre)allocated space bytes
      "numExtents" : 13,            // number of extents
      "nindexes" : 1,               // number of indexes
      "lastExtentSize" : 68579328,  // size created extent in bytes
      "paddingFactor" : 1,
      "systemFlags" : 1,
      "userFlags" : 1,
      "totalIndexSize" : 20211072,  // total index size in bytes
      "indexSizes" : {              // size of specific indexes bytes
            "_id_" : 20211072
      },
      "ok" : 1
}
```

Number of unique users in the node and way elements

```
db.alaska.distinct( 'created.user' ).length
```

343

I also ran the count for unique users in the node and way element in python and obtained the same count of 343.

Additional Check on the count of unique users

```
db.alaska.aggregate(
     {
          $group:
               {
                    _id:"$created.user"
               }
     },
     {
          $group:
               {
                    _id:null, "count":{$sum:1}
               }
     }
)
{ "_id" : null, "count" : 343 }
```

Number of Nodes

```
db.alaska.find({"type":"node"}).count()
585732
```

Number of ways

```
db.alaska.find({"type":"way"}).count()
36217
```

Top 15 contributing users sorted by total contributions

```
db.alaska.aggregate({$group:{_id:"$created.user",
"count":{"$sum":1}}}, {$sort: {count:-1}}, {$limit:15})
```

```
{ "_id" : "AlaskaDave", "count" : 160131 }
{ "_id" : "Brad Meteor", "count" : 69964 }
{ "_id" : "Jason Reid", "count" : 65811 }
{ "_id" : "Chris Lawrence", "count" : 50915 }
{ "_id" : "rsarwas", "count" : 35054 }
{ "_id" : "woodpeck_fixbot", "count" : 32486 }
{ "_id" : "dmgroom_ct", "count" : 27041 }
{ "_id" : "dmgroom_coastlines", "count" : 16169 }
{ "_id" : "jot", "count" : 14034 }
{ "_id" : "ke7diz", "count" : 13925 }
{ "_id" : "OJW", "count" : 12099 }
{ "_id" : "AndrewBuck", "count" : 11474 }
{ "_id" : "bbmiller", "count" : 7459 }
{ "_id" : "balrog-kun", "count" : 7236 }
{ "_id" : "greggerm", "count" : 5901 }
```

Number of users appearing with one post (having 1 post)

```
db.alaska.aggregate(
	{
		$group:
			{
				_id:"$created.user", "count":{$sum:1}
			}
	},
	{
		$match:{ "count" : {$lt:2}}
	},
	{
		$group:
			{
				_id:"Total", "users":{$sum:1}
			}
	}
)
```

```
{ "_id" : "Total", "users" : 63 }
```

How often is the map updated? Average per day per year results.

```
db.alaska.aggregate(
	{
		$project:
			{
			year: { $substr: [ "$created.timestamp", 0, 4 ] },
			month: { $substr: [ "$created.timestamp", 5, 2 ] },
			day: { $substr: [ "$created.timestamp", 8, 2 ] },
			hour: { $substr: [ "$created.timestamp", 11, 2 ] }
			}
	},
	{
		$group:
			{
				_id: {year:"$year", day:"$day"}, edits:{$sum:1}
			}
	},
	{
		$group:
			{
				_id: {year:"$_id.year"}, avgperday : { $avg:
"$edits" }
			}
	},
	{
		$sort:{avgperday:-1}
	}
)
```

```
{ "_id" : { "year" : "2015" }, "avgperday" : 177.33333333333334 }
{ "_id" : { "year" : "2014" }, "avgperday" : 4548.903225806452 }
{ "_id" : { "year" : "2013" }, "avgperday" : 4518.290322580645 }

{ "_id" : { "year" : "2012" }, "avgperday" : 2419.2 }
{ "_id" : { "year" : "2011" }, "avgperday" : 1762.5714285714287 }
{ "_id" : { "year" : "2010" }, "avgperday" : 1025.9354838709678 }

{ "_id" : { "year" : "2009" }, "avgperday" : 3483.3 }
{ "_id" : { "year" : "2008" }, "avgperday" : 3051 }
{ "_id" : { "year" : "2007" }, "avgperday" : 851.75 }
```

We can see the drop in edits or adds during 2010 to 2012, with the work picking back up in 2013.

Top 15 amenities

```
db.alaska.aggregate(
	{
		$match:{ amenity : {$exists:1}}
	},
	{
		$group:{_id:"$amenity","count":{$sum:1}}
	},
	{
		$sort:{count:-1}
	},
	{
		$limit:15
	}
)
```

```
{ "_id" : "parking", "count" : 547 }
{ "_id" : "school", "count" : 92 }
{ "_id" : "restaurant", "count" : 67 }
{ "_id" : "fuel", "count" : 52 }
{ "_id" : "fast_food", "count" : 38 }
{ "_id" : "toilets", "count" : 36 }
{ "_id" : "place_of_worship", "count" : 33 }
{ "_id" : "cafe", "count" : 32 }
{ "_id" : "fire_station", "count" : 16 }
{ "_id" : "post_office", "count" : 13 }
{ "_id" : "bank", "count" : 11 }
{ "_id" : "bar", "count" : 10 }
{ "_id" : "police", "count" : 9 }
{ "_id" : "hospital", "count" : 9 }
{ "_id" : "shelter", "count" : 8 }
```

Other ideas about the datasets

The data in OpenStreet maps can be used in many ways in other projects. Examining the population count and areas from Census data and combining this data with the amenities per population.

Also, using the node/way points with the changeset to monitor the growth of the objects on the map to see if the number of objects increases as population increases vs. what kind of infrastructure is being added by the city. If the OpenStreet map data falls behind we can also monitor the data to see how long the latency is between the new infrastructure added and the point added to open street maps.

Additional ways of improving and analyzing the data

1. Examine all the keys in the node and way elements before assuming to process the elements with a specific algorithm. The reason I state this is because we treated anything key with more than one colon exactly the same as any other key by dropping the text on the left of the first colon.

   The are many <tag k="x:y:z", v="something" /> like this in the dataset which are machine processed and uploaded by users. Most of these keys seem to be generated by GPS devices such as Garmin and Tiger.

2. For US Postal Codes, we need to have a database of zip codes and the city and state each zip code. When a zip code is processed the city and state can be reviewed, and checked against the expected city or state the zip code belongs. One could even validate the zip code, and then update the city and state based on the correct zip code and disregard the actual city and state entered by the user and use the city and state from the database.

   If we assume the zip code is correct and the zip code is incorrect or off by a digit due to manual entry we could accidently have the wrong city or state applied to the element in the dataset.

   We also have ZipLeft and ZipRight in the dataset, which needs to be cleaned and mapped into the address object.

- Latitude and Longitude needs to be validated. This can be accomplished by checking to see if the Lat and Lon fall within the boxed range we collected from the map bounds.

```
<bounds
     minlat="60.495"
     minlon="-151.359"
     maxlat="61.669"
     maxlon="-148.876"
/>
```

Additional Data Exploration

The most edits per month by user

```
db.alaska.aggregate(
	{
		$project:
			{
			user: "$created.user",
			year: { $substr: [ "$created.timestamp", 0, 4 ] },
			month: { $substr: [ "$created.timestamp", 5, 2 ] },
			day: { $substr: [ "$created.timestamp", 8, 2 ] },
			hour: { $substr: [ "$created.timestamp", 11, 2 ] }
			}
	},
	{
		$group:
			{
				_id: {user: "$user", year:"$year",
month:"$month"}, edits:{$sum:1}
			}
	},
	{
		$sort:{edits:-1}
	},
	{
		$limit:10
	}
)
{ "_id" : { "user" : "Brad Meteor", "year" : "2013", "month" : "12" }, "edits" : 69964 }
{ "_id" : { "user" : "Jason Reid", "year" : "2008", "month" : "01" }, "edits" : 65811 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "03" }, "edits" : 60085 }
{ "_id" : { "user" : "Chris Lawrence", "year" : "2009", "month" : "04" }, "edits" : 50915 }
{ "_id" : { "user" : "dmgroom_ct", "year" : "2011", "month" : "04" }, "edits" : 18851 }
{ "_id" : { "user" : "dmgroom_coastlines", "year" : "2011", "month" : "04" }, "edits" : 16169 }
{ "_id" : { "user" : "woodpeck_fixbot", "year" : "2009", "month" : "09" }, "edits" : 15087 }
{ "_id" : { "user" : "rsarwas", "year" : "2009", "month" : "11" }, "edits" : 11936 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "04" }, "edits" : 11425 }
{ "_id" : { "user" : "jot", "year" : "2012", "month" : "04" }, "edits" : 11079 }
```

Total edits per month and year.... for AlaskaDave sorted by total edits.

```
db.alaska.aggregate(
	{$match:{"created.user":"AlaskaDave"}}
	,
	{
		$project:
			{
			user: "$created.user",
			year:  { $substr: [ "$created.timestamp", 0, 4 ] },
			month: { $substr: [ "$created.timestamp", 5, 2 ] },
			day:   { $substr: [ "$created.timestamp", 8, 2 ] },
			hour:  { $substr: [ "$created.timestamp", 11, 2 ] }
			}
	},
	{
		$group:
			{
				_id: {user: "$user",
					year:"$year",
					month:"$month"},
				edits:{$sum:1}
			}
	},
	{
		$sort:{edits:-1}
	}
)
```


{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "03" }, "edits" : 60085 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "04" }, "edits" : 11425 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "08" }, "edits" : 10571 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2012", "month" : "12" }, "edits" : 9701 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "06" }, "edits" : 9034 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "09" }, "edits" : 8429 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "05" }, "edits" : 8163 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "07" }, "edits" : 7644 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "04" }, "edits" : 7409 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "06" }, "edits" : 4807 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "01" }, "edits" : 3702 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "07" }, "edits" : 3665 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "10" }, "edits" : 2990 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "02" }, "edits" : 2409 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "01" }, "edits" : 1762 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "03" }, "edits" : 1673 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "02" }, "edits" : 1202 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "05" }, "edits" : 1136 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "12" }, "edits" : 1127 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2012", "month" : "11" }, "edits" : 978 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "10" }, "edits" : 862 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "09" }, "edits" : 486 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2014", "month" : "11" }, "edits" : 318 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "11" }, "edits" : 267 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "12" }, "edits" : 243 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2013", "month" : "08" }, "edits" : 36 }
{ "_id" : { "user" : "AlaskaDave", "year" : "2011", "month" : "06" }, "edits" : 6 }

The number of missing Lat and Lon

A good position consists of any value not equal to zero. We are assuming Alaska is not on the prime meridian.

good -> "pos" : [61.1808649,-149.8862164]

bad -> "pos" : [0,0]

```
db.alaska.aggregate(
	{
		$match:{ pos: {$exists:1}}
	},
	{
		$unwind : "$pos"
	},
	{
		$match : {pos: {$ne:0}}
	},
	{
		$group:
		{
			_id:"$_id","count":{$sum:1}}
	},
	{
		$group:
		{
			_id:"No_Position_Values","count":{$sum:1}
		}
	}
)
```

{ "_id" : "No_Position_Values", "count" : 586343 }

Check to make sure the ObjectID's with no position values have a count of 1. We need this because of the 'unwind' which creates two records for every ObjectID with a Pos[] key. If we have a count less than 2, this means one value of the Pos [] is a zero and we have one good and one bad value.

```
db.alaska.aggregate(
	{
		$match:{ pos: {$exists:1}}
	},
	{
		$unwind : "$pos"
	},
	{
		$match : {pos: 0}
	},
	{
		$group:
		{_id:"$_id","count":{$sum:1}}
	},
	{$match : {count : {$lt: 2}}}
)
```

0 Results

Check for Lat and Lon, by using max and min values. Typically any lat and lon should be within +180 and -180 degrees.

```
db.alaska.aggregate(
     {
          $match:{ pos: {$exists:1}}
     },
     {
          $unwind : "$pos"
     },
     {
          $group:
          {
               _id: "$item",
               minPos: { $min: "$pos" },
               maxPos: { $max: "$pos" }
          }
     }
)
```

{ "_id" : null, "minPos" : -154.933694, "maxPos" : 64.8011404 }

The map bounds are

```
<bounds
     minlat="58.757" minlon="-152.831"
     maxlat="62.047" maxlon="-145.92"/>
```

Based on the map bounds we have at least one point out of the map bounds as pulled using the API.

```
db.alaska.aggregate( { $match:{ pos: {$exists:1}} }, {
$unwind : "$pos"  }, { $match:{ "pos" : {$lt:-154}} }
).pretty()

//* Copy the first id and use this to query the element for
review
> db.alaska.find({"id":"2039551277"}).pretty()
{
     "_id" : ObjectId("54f387b031c54d52d7e9bf00"),
     "id" : "2039551277",
     "type" : "node",
     "pos" : [
          56.443474,
          -154.233608
     ],
     "created" : {
          "changeset" : "14082505",
          "user" : "pnorman_imports",
          "version" : "1",
          "uid" : "902766",
          "timestamp" : "2012-11-29T07:32:22Z"
     }
}
As we can see, this record falls outside of the map bounds.
```

To 10 users who are the big offenders for not adding in Position (Lat & Lon)

```
db.alaska.aggregate(
      {
            $match:{ pos: {$exists:1}}
      },
      {
            $unwind : "$pos"
      },
      {
            $match : {pos: {$ne:0}}
      },
      {
            $group:
            {
                  _id:"$created.user","count":{$sum:1}}
      },
      {
            $sort: {count:-1}
      },
      {
            $limit: 10
      }
)
{ "_id" : "AlaskaDave", "count" : 305420 }
{ "_id" : "Brad Meteor", "count" : 134924 }
{ "_id" : "Jason Reid", "count" : 129788 }
{ "_id" : "Chris Lawrence", "count" : 101830 }
{ "_id" : "rsarwas", "count" : 67484 }
{ "_id" : "woodpeck_fixbot", "count" : 64972 }
{ "_id" : "dmgroom_ct", "count" : 51332 }
{ "_id" : "dmgroom_coastlines", "count" : 31450 }
{ "_id" : "jot", "count" : 27564 }
{ "_id" : "ke7diz", "count" : 27246 }
```

AlaskaDave has the most empty sets of Positions, and also has the mode total edits in this dataset.

How many schools are there in this area have Lat and Lon?

```
db.alaska.aggregate(
      {
            $match:{ amenity : {$exists:1}}
      },
      {
            $match:{ pos: {$exists:1}}
      },
      {
            $match:{ name: {$exists:1}}
      },
      {
            $match:{ amenity : "school"}
      },
      {
            $unwind : "$pos"
      },
      {
            $match : {pos: {$ne:0}}
      },
      {
            $project:
                  {
                    school:"$name",
                    location: "$pos",
                    address: "$address",
                    _id: 0
                  }
      },
      {
            $group:
                  {
                        _id:null, "count":{$sum:1}
                  }
      })
```

We have a total of { "_id" : "school", "count" : 66 }
Since we have 2 values for Lat and Lon the total schools with a Lat and Lon value not equal to 0 is 33 schools.

```
db.alaska.aggregate(
     {
          $match:{ amenity : {$exists:1}}
     },
     {
          $match:{ pos: {$exists:1}}
     },
     {
          $match:{ name: {$exists:1}}
     },
     {
          $match:{ amenity : "school"}
     },
     {
          $unwind : "$pos"
     },
     {
          $match : {pos:0}
     },
     {
          $project:
               {
                 school:"$name",
                 location: "$pos",
                 address: "$address",
                 _id: 0
               }
     },
     {
          $group:
               {
                    _id:null, "count":{$sum:1}
               }
     })
```

We have a total of { "_id" : "school", "count" : 76 }

Since we have 2 values for Lat and Lon the total schools with a Lat and Lon value matching to 0 is 38 schools.

Total Schools

```
db.alaska.aggregate(
      {
            $match:{ amenity : {$exists:1}}
      },
      {
            $match:{ amenity : "school"}
      },
      {
            $match:{ name: {$exists:1}}
      },
      {
            $group:
                  {
                        _id:null, "count":{$sum:1}
                  }
      })
```

We have a total of { "_id" : "school", "count" : 71 }

What is the count of cities existing in this area?

```
db.alaska.aggregate(
      {
            $match:{ "address.city" : {$exists:1}}
      },
      {
            $group:
            {
                  _id:"$address.city","count":{$sum:1}}
      },
      {
            $sort:{count:-1}
      }
)
```

```
{ "_id" : "Anchorage", "count" : 50 }
{ "_id" : "Homer", "count" : 7 }
{ "_id" : "Wasilla", "count" : 3 }
{ "_id" : "Palmer", "count" : 2 }
{ "_id" : "Soldotna", "count" : 2 }
{ "_id" : "Eagle River", "count" : 1 }
{ "_id" : "Girdwood", "count" : 1 }
{ "_id" : "Whittier", "count" : 1 }
{ "_id" : "Kasilof", "count" : 1 }
{ "_id" : "He", "count" : 1 }
{ "_id" : "Kenai", "count" : 1 }
```

What type of land use is found the most?

```
db.alaska.aggregate(
      {
            $match:{ landuse : {$exists:1}}
      },
      {
            $group:
            {
                  _id:"$landuse","count":{$sum:1}}
      },
      {
            $sort:{count:-1}
      }
)
```

```
{ "_id" : "quarry", "count" : 102 }
{ "_id" : "forest", "count" : 64 }
{ "_id" : "industrial", "count" : 48 }
{ "_id" : "meadow", "count" : 30 }
{ "_id" : "commercial", "count" : 25 }
{ "_id" : "retail", "count" : 20 }
{ "_id" : "residential", "count" : 14 }
{ "_id" : "recreation_ground", "count" : 7 }
{ "_id" : "grass", "count" : 4 }
{ "_id" : "military", "count" : 3 }
{ "_id" : "reservoir", "count" : 2 }
{ "_id" : "brownfield", "count" : 2 }
{ "_id" : "landfill", "count" : 2 }
{ "_id" : "cemetery", "count" : 2 }
{ "_id" : "farmland", "count" : 1 }
{ "_id" : "plant_nursery", "count" : 1 }
{ "_id" : "allotments", "count" : 1 }
{ "_id" : "village_green", "count" : 1 }
{ "_id" : "construction", "count" : 1 }
{ "_id" : "railway", "count" : 1 }
{ "_id" : "basin", "count" : 1 }
{ "_id" : "gravel_pit", "count" : 1 }
```

References

Mongodb Documentation

Retrieved From: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/

OpenStreet Map Documentation

Retrieved From:
http://www.openstreetmap.org/
http://wiki.openstreetmap.org/wiki/Overpass_API
http://wiki.openstreetmap.org/wiki/API_v0.6/XSD
http://wiki.openstreetmap.org/wiki/User:AlaskaDave

Average Latitude and Longitude

http://dev.maxmind.com/geoip/legacy/codes/state_latlon/

About AlaskaDave

http://wiki.openstreetmap.org/wiki/User:AlaskaDave

User:AlaskaDave
I am retired and split my time between Chiang Mai, Thailand and Homer, Alaska. I am a very active mapper in both regions. My other pastimes are tennis, photography and motorcycling. I have a travel blog, I'm Outta Here, [1] where I write about my travels. I started mapping a couple of years ago and it has become practically an obsession. I started out using Potlatch-2 but quickly moved to JOSM for it's greater power and customizability.

The other areas I've actively contributed to are Buffalo, NY, and surrounding areas, the Adirondack Park in northern NY State, along with Eugene, Oregon, and Greensboro, NC where I have relatives. One of my long term goals is to add to OSM all the tennis courts in the areas I frequently visit.

I've completely updated and corrected the Tiger street data for Homer and surrounds and recently did a tour of central Alaska on the Denali Highway and have done a ton of work on both highway and river systems along the route (Glenn, Denali, Richardson, and Parks Highways). Alaska is vast and there are few people mapping it so there is plenty of opportunity for others to get involved and to make a difference.

Installing Mongodb

http://docs.mongodb.org/manual/tutorial/install-mongodb-on-os-x/

$ brew update
$ brew install mongodb
$ brew unlink mongodb
$ mkdir -p /data/db
$ sudo chown `id -u` /data/db

$ ls -p /data/db
_tmp/ journal/ local.0 local.ns mongod.lock

# Test the mongodb instance
$ mongod

mongo
MongoDB shell version: 2.6.7
connecting to: test

mongod

```
cn $ cd
/Users/cn/Dropbox/Udacity_DataScience/P2/Lesson_6/PreparingForDatabase

cn PreparingForDatabase $ mongo
```

```
MongoDB shell version: 2.6.7
connecting to: test
> use osm
switched to db osm
> ^C

cn PreparingForDatabase $  mongoimport --db osm --collection alaska --
file Alaska.xml.json
connected to: 127.0.0.1
2015-02-21T13:57:27.000-0600        Progress: 17400896/138627567  12%
2015-02-21T13:57:27.000-0600              86900 28966/second
2015-02-21T13:57:30.000-0600        Progress: 36657219/138627567  26%
2015-02-21T13:57:30.000-0600              184400      30733/second
2015-02-21T13:57:33.000-0600        Progress: 55824369/138627567  40%
2015-02-21T13:57:33.000-0600              280300      31144/second
2015-02-21T13:57:36.019-0600        Progress: 75763207/138627567  54%
2015-02-21T13:57:36.019-0600              379900      31658/second
2015-02-21T13:57:39.001-0600        Progress: 95458070/138627567  68%
2015-02-21T13:57:39.001-0600              477100      31806/second
2015-02-21T13:57:42.000-0600        Progress: 113447463/138627567 81%
2015-02-21T13:57:42.000-0600              566000      31444/second
2015-02-21T13:57:44.397-0600 check 9 622579
2015-02-21T13:57:44.423-0600 imported 622579 objects
```

“I hereby confirm that this submission is my work. I have cited above the origins of any parts of the submission that were taken from Websites, books, forums, blog posts, github repositories, etc. By including this in my email, I understand that I will be expected to explain my work in a video call with a Udacity coach before I can receive my verified certificate.”