Santa Cruz County OpenStreetMap Data

October 3, 2016

1 Introduction

For this project, we will be auditing and cleaning OpenStreetMap data for Santa Cruz County in California. During the course, we looked at OSM data for Chicago and found inconsistencies in street name abbreviations and postal codes. We will look for the same problems in the Santa Cruz data. After cleaning it up, we will rewrite the data as a JSON file and upload it to MongoDB. We can use MongoDB to explore the data and make any more changes if necessary.

2 Auditing the Data

2.1 Street Name Abbreviations

We don't want any abbreviated street names in our data. We expect the last word of each street name to be Street, Avenue, Road, etc. We'll parse through the street names in this set and create a dictionary of unexpected and abbreviated values.

```
In [2]: import xml.etree.cElementTree as ET
        from collections import defaultdict
        import re
        import pprint

        OSMFILE = "santa-cruz_california.osm"
        street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

        # Acceptable street types
        expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square",
                    "Lane", "Road", "Trail", "Parkway", "Commons", "Way"]

        # Search for unexpected street types and add to a dictionary
        def audit_street_type(street_types, street_name):
            m = street_type_re.search(street_name)
            if m:
                street_type = m.group()
                if street_type not in expected:
                    street_types[street_type].add(street_name)

        # Check if element key is addr:street
        def is_street_name(elem):
            return (elem.attrib['k'] == "addr:street")

        # Puts all above pieces together
        def audit(osmfile):
            osm_file = open(osmfile, "r")
            street_types = defaultdict(set)
            for event, elem in ET.iterparse(osm_file, events=("start",)):

                if elem.tag == "node" or elem.tag == "way":
                    for tag in elem.iter("tag"):
                        if is_street_name(tag):
                            audit_street_type(street_types, tag.attrib['v'])

            return street_types

        st_types = audit(OSMFILE)
        pprint.pprint(dict(st_types))
```

```
{'245': set(['245']),
 '9': set(['Highway 9', 'Hwy 9', 'Hwy. 9']),
 'Ave': set(['220 Sylvania Ave', '41st Ave', 'Chanticleer Ave']),
 'Cedar': set(['Cedar']),
 'Chestnut': set(['Chestnut']),
 'Circle': set(['Baskin Circle']),
 'Esplanade': set(['Esplanade']),
 'Extension': set(['Mission Street Extension']),
 'Front': set(['Front']),
 'Gulch': set(['Rodeo Creek Gulch']),
 'Loop': set(['Engineering Loop']),
 'Mar': set(['Rancho Del Mar']),
 'Merrill': set(['Merrill']),
 'Pacific': set(['Pacific']),
 'Ramps': set(['@ Pasatiempo Sb Ramps']),
 'Rd': set(['Mount Hermon Rd']),
 'Seabright': set(['Seabright']),
 'St': set(['225 Rooney St']),
 'front': set(['front'])}
```

There are a few Avenues, Roads, and Streets written as Ave, Rd, and St respectively. To make the data consistent, we want to chage Ave to Avenue, Rd to Road, and St to Street. We also want to look up Cedar, Chestnut, Front, Merrill, Pacific, and Seabright on another map and see if they are Streets, Avenues, or Roads and change them accordingly.

The rest of the unexpected street name endings seem legitimate except for the entry that is just 245. We can look at the full version of this document once we have our data in MongoDB. It is most likely supposed to be a housenumber or apartment number.

Highway 9 should be wirtten out consistently. Right now it is written in three different ways. We will perform these tasks as we write the JSON version of the file a little later.

2.2 Postal Code Inconsistencies

We now want to look at our set of postal codes. We can write a code similar to our street name audit to print the set of postal codes in our data. There is no need for a dictionary since we want to look at the full postal code entry. Instead we'll just create the set of all postal codes.

```
In [4]: def is_postcode(elem):
            return (elem.attrib['k'] == "addr:postcode")


        def audit_postcode(osmfile):
            osm_file = open(osmfile, "r")
```

```
    postcodes = set()
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_postcode(tag):
                    postcodes.add(tag.attrib['v'])
    return postcodes


pprint.pprint(audit_postcode(OSMFILE))
```

```
set(['1982',
     '95002',
     '95003',
     '95010',
     '95018',
     '95041',
     '95060',
     '95062',
     '95062-4205',
     '95064',
     '95065',
     '95065-1711',
     '95066',
     '95066-4024',
     '95066-5121',
     '95073',
     'CA 95062',
     'CA 95065'])
```

Some of the postal codes are in 9 digit form instead of the more common 5 digit form, and the last two of the set begin with CA. There is also an entry 1982, which isn't a postal code at all. It is hard to guess what the user meant without looking at the entire entry. I looked up all of the postal codes to make sure they are part of Santa Cruz County. The only out of place code is 95003, which is Alviso, CA, a town just north of San Jose, CA. We will look at the full entry in MongoDB to try to make sense of it.

2.3 Converting OSM to JSON

Before we upload the data to MongoDB, we have to rewrite it in JSON form. We only want to process top-level tags "node" and "way", turning their attributes in key/value pairs, except: - attributes in the CREATED array should be added under a key "created" - attributes for latitude and longitude should be added to a "pos" array, for use in geospacial indexing. Make sure the values inside "pos" array are floats and not strings. - if second level tag "k" value contains problematic characters, it should be ignored - if second level tag "k" value starts with "addr:", it should be added to a dictionary "address" - if second level tag "k" value does not start with "addr:", but contains ":", you can process it same as any other tag. - if there is a second ":" that separates the type/direction of a street, the tag should be ignored. - for "way" specifically, tags starting with "nd ref" should be added to array called "node_refs".

We will also extend street name abbreviations and add Avenue or Street to the proper street names. Postal codes, aside from 1982, will be converted to a 5-digit form.

2.3.1 A Note on id's

When we upload our data into MongoDB, each entry will be assigned a unique id under '_id'. Our data already has an id field, but in order to use it we have to make sure that each 'id' is truely unique in our

OSM file. If so, we'll have to match the id value to the '_id' key instead of a 'id' key. If we try to write code to check for uniqueness, we'll essentially have to compare over 250,000 lines against each other. This would take a ton of time to compute so instead I looked up how id's in OSM are created and found this: http://gis.stackexchange.com/questions/103572/are-osm-ids-unique-over-all-object-types. One user claims that OSM id's are composed of type of object, id, version, and combination of tags. Though it is highly unlikely, there is still a possibility that a node and way have the same id number. The larger the file, the more likely it is that we run into a problem. We'll let MongoDB assign unique numbers to the '_id' field.

```
In [8]: import codecs
        import json

        # Maps endings to proper edits
        mapping = { "St": "Street",
                    "St.": "Street",
                    "Rd": "Road",
                    "Ave": "Avenue",
                    "Cedar": "Cedar Street",
                    "Chestnut": "Chestnut Street",
                    "front": "Front Street",
                    "Front": "Front Street",
                    "Merrill": "Merrill Street",
                    "Pacific": "Pacific Avenue",
                    "Seabright": "Seabright Avenue"
                  }


        def update_name(name, mapping):
            m = street_type_re.search(name)
            m = m.group()
            if m in mapping.keys():
                name = name.replace(m, mapping[m])

            return name

        # gets rid of 'CA' and converts 9 digit to 5 digit
        def update_postcode(postcode):
            if postcode.startswith('CA'):
                postcode = postcode[3:]

            elif len(postcode) > 5:
                postcode = postcode[0:5]

            else:
                postcode = postcode

            return postcode


        lower = re.compile(r'^([a-z]|_)*$')
        lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
        problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

        # attributes in the CREATED array should be added under a key "created"
        CREATED = ["version", "changeset", "timestamp", "user", "uid"]
```

```
def shape_element(element):
    node = {}
    if element.tag == "node" or element.tag == "way" :
        # keep track of type
        node['type'] = element.tag
        node['created'] = {}
        for key in element.attrib.keys():
            # attributes of "node" and "way" should be turned into regular key/value
            # pairs
            # attributes in the CREATED array should be added under a key "created"
            if key in CREATED:
                node['created'][key] = element.attrib[key]

            elif key not in ['lat', 'lon']:
                node[key] = element.attrib[key]

        # attributes for latitude and longitude should be added to a "pos" array
        # not all entries will have latitude and longitude
        try:
            node['pos'] = [float(element.attrib['lat']), float(element.attrib['lon'])]
        except:
            pass

        # if second level tag "k" value contains problematic characters, it should be
        # ignored
        for child in element:

            if child.tag == 'tag':
                key = child.attrib['k']
                value = child.attrib['v']
                if re.search(problemchars, key):
                    continue
                # if second level tag "k" value starts with "addr:", it should be added
                # to a dictionary "address"
                if key.startswith('addr:'):
                    # if there is a second ":" that separates the type/direction of a
                    # street, the tag should be ignored
                    if re.search(lower_colon, key[5:]):
                        continue

                    # update street names using mapping if necessary
                    elif key == 'addr:street':
                        new_value = update_name(value, mapping)
                        if 'address' in node:
                            node['address']['street'] = new_value
                        else:
                            node['address'] = {}
                            node['address']['street'] = new_value

                    # fix postal codes
                    elif key == 'addr:postcode':
                        new_value = update_postcode(value)
                        if 'address' in node:
                            node['address']['postcode'] = new_value
                        else:
                            node['address'] = {}
                            node['address']['postcode'] = new_value

                    else:
                        if 'address' in node:
                            node['address'][key[5:]] = value
                        else:
                            node['address'] = {}
                            node['address'][key[5:]] = value

                # if second level tag "k" value does not start with "addr:", but
                # contains ":", you can process it same as any other tag.
                else:
                    node[key] = value

            # second level 'nd' tags should be made into an array
            elif child.tag == 'nd':
                if 'node_refs' in node:
                    node['node_refs'].append(child.attrib['ref'])
                else:
                    node['node_refs'] = []
                    node['node_refs'].append(child.attrib['ref'])

        return node
    else:
        return None


def process_map(file_in, pretty = False):
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return data

# NOTE: if you are running this code on your computer, with a larger dataset,
# call the process_map procedure with pretty=False. The pretty=True option adds
# additional spaces to the output, making it significantly larger.
data = process_map('santa-cruz_california.osm', False)
```

This completes auditing and cleaning the data for now. We can now upload the data to MongoDB and explore the data. If necessary, more cleaning can be done there.

3 Cleaning the Data in MongoDB

We have uploaded the data as a collection santacruz into a database called maps. There are two options for looking at the data, either through the command line or with python using the pymongo library. Since we are working in an iPython Notebook we will use pymongo. The commands are very similar if not the same.

We have three things we have to investigate or fix. We must - look at and update the entry with street name 245. - write Highway 9 consistently. - fix entry with postal code 1982.

3.1 Street Name 245

Next we will look at street name '245' and try to figure out the user's intentions when making this entry.

```
In [34]: strange_street = db.santacruz.find({'address.street':'245'})
         pprint.pprint(list(strange_street))
```

```
[{u'_id': ObjectId('5647e57367df1b2b7ef6d90e'),
  u'address': {u'street': u'245', u'unit': u'V'},
  u'created': {u'changeset': u'19816930',
               u'timestamp': u'2014-01-05T03:32:28Z',
               u'uid': u'173918',
               u'user': u'njaard',
               u'version': u'1'},
  u'id': u'2610436134',
  u'name': u'Cutesy Cupcakes',
  u'pos': [37.046047, -122.0300636],
  u'shop': u'confectionery',
  u'type': u'node'}]
```

In the address dictionary, we have street 245 and unit V. There are two clues to what the address of this document should be. We have a name, Cutesy Cupcakes, and the coordinates listed under the 'pos' key. Since it makes most sense that 245 is the building number, I googled 'Cutesy Cupcakes 245v' and came accross two different addresses. One lists Cutesy Cupcakes at 235 Mt Hermon Road Suite A in Scotts Valley, while the other is right next door at 245V Mt Hermon Road. Yelp and Google Maps list Cutesy Cupcakes at 235 Mt Hermon Road. I think it is safe to go with Yelp and Google Maps. We will also update city and postal code.

So far we have been cleaning away abbreviated street names. It makes sense to keep this trend and call the street Mount Hermon Road instead of Mt Hermon Road.

```
In [36]: db.santacruz.update_one(
             {
                 'address.street':'245'
             },
             {
                 '$set':{'address':
                         {
                         'housenumber':'245',
                         'street':'Mount Hermon Road',
                         'unit':'A',
                         'city':'Scotts Valley',
                         'state':'CA',
                         'postcode':'95066'
                         }
                        }
             }
         )
```

```
        updated_entry = db.santacruz.find({'id':'2610436134'})

        pprint.pprint(list(updated_entry))
```

```
[{u'_id': ObjectId('5647e57367df1b2b7ef6d90e'),
  u'address': {u'city': u'Scotts Valley',
               u'housenumber': u'245',
               u'postcode': u'95066',
               u'state': u'CA',
               u'street': u'Mount Hermon Road',
               u'unit': u'A'},
  u'created': {u'changeset': u'19816930',
               u'timestamp': u'2014-01-05T03:32:28Z',
               u'uid': u'173918',
               u'user': u'njaard',
               u'version': u'1'},
  u'id': u'2610436134',
  u'name': u'Cutesy Cupcakes',
  u'pos': [37.046047, -122.0300636],
  u'shop': u'confectionery',
  u'type': u'node'}]
```

3.2 Highway 9 Problem

Let's make all streets called Highway 9 look the same. We know there is at least one document with street Hwy 9 and one with street Hwy. 9. Below we search for those entries and while they are on our screen we will make sure there is no reason for these to be named this way.

```
In [30]: from pymongo import MongoClient
         import pprint

         client = MongoClient()

         db = client['maps']

         wrong_hwy_nine = db.santacruz.find({'address.street':{'$in':['Hwy 9', 'Hwy. 9']}})

         pprint.pprint(list(wrong_hwy_nine))
```

```
[{u'_id': ObjectId('5647e57267df1b2b7ef6bfdd'),
  u'address': {u'city': u'Felton',
               u'housenumber': u'5447',
               u'postcode': u'95018',
               u'street': u'Hwy. 9'},
  u'amenity': u'restaurant',
  u'created': {u'changeset': u'16489935',
               u'timestamp': u'2013-06-10T00:55:29Z',
               u'uid': u'751955',
               u'user': u'peterm95018',
               u'version': u'2'},
  u'cuisine': u'italian',
  u'id': u'2338593307',
  u'name': u'Oak Tree Ristorante',
  u'outdoor_seating': u'yes',
  u'phone': u'831-335-5551',
  u'pos': [37.040803, -122.072386],
  u'takeaway': u'yes',
  u'type': u'node',
  u'website': u'Http://oaktreetistorante.com'},
 {u'_id': ObjectId('5647e57267df1b2b7ef6c3e7'),
  u'address': {u'city': u'Felton',
               u'housenumber': u'6205',
               u'postcode': u'95018',
               u'street': u'Hwy 9'},
  u'amenity': u'restaurant',
  u'created': {u'changeset': u'18825021',
               u'timestamp': u'2013-11-10T21:25:10Z',
               u'uid': u'173918',
               u'user': u'njaard',
               u'version': u'3'},
  u'cuisine': u'pizza',
  u'id': u'2371463445',
  u'name': u'Redwood Pizza',
  u'outdoor_seating': u'yes',
  u'pos': [37.0514423, -122.0734682],
  u'smoking': u'no',
  u'takeaway': u'yes',
  u'type': u'node'}]
```

Nothing peculiar is going on here. The street name was just written in a different manner, so we can go ahead and update both of these entries.

```
In [32]: db.santacruz.update_many(
             {
                 'address.street':{'$in':['Hwy 9', 'Hwy. 9']}
             },
             {
                 '$set':{'address.street':'Highway 9'}
             }
         )

         # look up entries to double check work
         updated_entries = db.santacruz.find({'id':{'$in':['2338593307', '2371463445']}})
         pprint.pprint(list(updated_entries))
```

```
[{u'_id': ObjectId('5647e57267df1b2b7ef6bfdd'),
  u'address': {u'city': u'Felton',
               u'housenumber': u'5447',
               u'postcode': u'95018',
               u'street': u'Highway 9'},
  u'amenity': u'restaurant',
  u'created': {u'changeset': u'16489935',
               u'timestamp': u'2013-06-10T00:55:29Z',
               u'uid': u'751955',
               u'user': u'peterm95018',
               u'version': u'2'},
  u'cuisine': u'italian',
  u'id': u'2338593307',
  u'name': u'Oak Tree Ristorante',
  u'outdoor_seating': u'yes',
  u'phone': u'831-335-5551',
  u'pos': [37.040803, -122.072386],
  u'takeaway': u'yes',
  u'type': u'node',
  u'website': u'Http://oaktreetistorante.com'},
 {u'_id': ObjectId('5647e57267df1b2b7ef6c3e7'),
  u'address': {u'city': u'Felton',
               u'housenumber': u'6205',
               u'postcode': u'95018',
               u'street': u'Highway 9'},
  u'amenity': u'restaurant',
  u'created': {u'changeset': u'18825021',
               u'timestamp': u'2013-11-10T21:25:10Z',
               u'uid': u'173918',
               u'user': u'njaard',
               u'version': u'3'},
  u'cuisine': u'pizza',
  u'id': u'2371463445',
  u'name': u'Redwood Pizza',
  u'outdoor_seating': u'yes',
  u'pos': [37.0514423, -122.0734682],
  u'smoking': u'no',
  u'takeaway': u'yes',
  u'type': u'node'}]
```

3.3 Postal Code Problem

After cleaning the postal codes earlier, we were still left with one odd document. Let's bring up the document with 1982 as a postal code.

```
In [19]: pprint.pprint(list(db.santacruz.find({'address.postcode':'1982'})))
```

```
[{u'_id': ObjectId('5647e57367df1b2b7ef6f739'),
  u'address': {u'city': u'Santa Cruz',
               u'housenumber': u'1502',
               u'postcode': u'1982',
               u'street': u'Pacific Avenue'},
  u'created': {u'changeset': u'22074373',
               u'timestamp': u'2014-05-01T22:34:11Z',
               u'uid': u'2032478',
               u'user': u'SarahStierch',
               u'version': u'2'},
  u'historic': u'building',
  u'historic:url': u'http://pdfhost.focus.nps.gov/docs/NRHP/Text/82002273.pdf',
  u'id': u'2830369542',
  u'name': u'Bank of Santa Cruz County',
  u'pos': [36.9753582, -122.0252398],
  u'ref:nhrp': u'82002273',
  u'type': u'node'}]
```

This document is for Bank of Santa Cruz County found at 1502 Pacific Avenue. Some quick research on the address tells us that this building is now a clothing store, but the Bank of Santa Cruz County building is a historic landmark. The address is correct, but the postal code should be 95060.

```
In [33]: db.santacruz.update_one(
             {
                 'address.postcode':'1982'
             },
             {
                 '$set':{'address.postcode':'95060'}
             }
         )

         updated_entry = db.santacruz.find({'id':'2830369542'})
         pprint.pprint(list(updated_entry))
```

```
[{u'_id': ObjectId('5647e57367df1b2b7ef6f739'),
  u'address': {u'city': u'Santa Cruz',
               u'housenumber': u'1502',
               u'postcode': u'95060',
               u'street': u'Pacific Avenue'},
  u'created': {u'changeset': u'22074373',
               u'timestamp': u'2014-05-01T22:34:11Z',
               u'uid': u'2032478',
               u'user': u'SarahStierch',
               u'version': u'2'},
  u'historic': u'building',
  u'historic:url': u'http://pdfhost.focus.nps.gov/docs/NRHP/Text/82002273.pdf',
  u'id': u'2830369542',
  u'name': u'Bank of Santa Cruz County',
  u'pos': [36.9753582, -122.0252398],
  u'ref:nhrp': u'82002273',
  u'type': u'node'}]
```

4 Exploring the Data

4.1 Entry Types

```
In [15]: docs = db.santacruz.find().count()
         nodes = db.santacruz.find({'type':'node'}).count()
         ways = db.santacruz.find({'type':'way'}).count()

         print 'Number of documents:', docs
         print '\nNumber of nodes:', nodes
         print '\nNumber of ways:', ways
```

```
Number of documents: 278190

Number of nodes: 257306

Number of ways: 20804
```

4.2 Users

```
In [14]: users = db.santacruz.distinct('created.user')

         top_users = db.santacruz.aggregate([
             {
                 '$group': {'_id':'$created.user', 'contributions':{'$sum':1}}
             },
             {
                 '$sort':{'contributions':-1}
             },
             {
                 '$limit':10
             }
             ])

         print 'Unique users:', len(users)
         print '\nTop 10 contributing users'
         for doc in top_users:
             print doc
```

```
Unique users: 395

Top 10 contributing users
{u'_id': u'stevea', u'contributions': 158863}
{u'_id': u'nmixter', u'contributions': 41746}
{u'_id': u'DanHomerick', u'contributions': 25696}
{u'_id': u'Apo42', u'contributions': 5822}
{u'_id': u'adelman', u'contributions': 5339}
{u'_id': u'woodpeck_fixbot', u'contributions': 4546}
{u'_id': u'Chris Lawrence', u'contributions': 3017}
{u'_id': u'hrutten', u'contributions': 2246}
{u'_id': u'RichRico', u'contributions': 1679}
{u'_id': u'ZekeFarwell', u'contributions': 1403}
```

Note that user stevea has contributed 158863, or 57.1%, of the documents in our collection. According to his wiki, he is very involved in the OSM project in Northen California, specifially in Santa Cruz, Monterey Bay, Silicon Valley, and South Bay. Stevea is interested in imporoving parks, trails, and bike routes starting with Santa Cruz. Along with some professors, stevea hosts mapping parties at UC Santa Cruz. You can read about him on his OSM wiki: http://wiki.openstreetmap.org/wiki/User:Stevea.

4.3 Addresses

```
In [24]: postcodes = db.santacruz.aggregate([
                {
                    '$match':{'address.postcode':{'$exists':1}}
                },
                {
                    '$group':{'_id':'$address.postcode', 'count':{'$sum':1}}
                },
                {
                    '$sort':{'count':-1}
                }
            ])

        print "Postal codes"
        for doc in postcodes:
            print doc
```

```
Postal codes
{u'count': 288, u'_id': u'95064'}
```

```
{u'count': 36, u'_id': u'95060'}
{u'count': 12, u'_id': u'95062'}
{u'count': 10, u'_id': u'95018'}
{u'count': 9, u'_id': u'95066'}
{u'count': 7, u'_id': u'95003'}
{u'count': 6, u'_id': u'95073'}
{u'count': 6, u'_id': u'95010'}
{u'count': 5, u'_id': u'95065'}
{u'count': 1, u'_id': u'95002'}
{u'count': 1, u'_id': u'95041'}
```

The overwhelming majority of documents do not include a postal code in the address section. Of those that do, most are coming from the UC Santa Cruz campus. Let's see if street names follow this trend.

```
In [23]: streetnames = db.santacruz.aggregate([
                 {
                     '$match':{'address.street':{'$exists':1}}
                 },
                 {
                     '$group':{'_id':'$address.street', 'count':{'$sum':1}}
                 },
                 {
                     '$sort':{'count':-1}
                 },
                 {
                     '$limit':15
                 }
             ])

         print "Street names"
         for doc in streetnames:
             print doc
```

```
Street names
{u'count': 35, u'_id': u'Porter-Kresge Road'}
{u'count': 20, u'_id': u'Village Road'}
{u'count': 18, u'_id': u'Stevenson Service Road'}
{u'count': 16, u'_id': u'East Road'}
{u'count': 16, u'_id': u'Mount Hermon Road'}
{u'count': 14, u'_id': u'Steinhart Way'}
{u'count': 14, u'_id': u'Pacific Avenue'}
{u'count': 14, u'_id': u'College Eight Service Road'}
{u'count': 14, u'_id': u'Cowell-Stevenson Road'}
{u'count': 13, u'_id': u'Heller Drive'}
{u'count': 13, u'_id': u'Emeline Avenue'}
{u'count': 11, u'_id': u'Merrill Road'}
{u'count': 10, u'_id': u'Crown Service Road'}
{u'count': 10, u'_id': u'College Ten Road'}
{u'count': 10, u'_id': u'Soquel Avenue'}
```

As expected, only three streets from the list above are off campus. Looking at this data I realize that we may have made a mistake earlier while cleaning the street names. I assumed that Merrill was meant to be Merrill Street in the actual town of Santa Cruz. Since Merrill Road is in the top 15 most common names, it is likely that Merrill was supposed to be Merrill Road. Let's take a look at Merril Street entries, and if necessary, we'll edit the street name.

```
In [21]: pprint.pprint(list(db.santacruz.find({'address.street':'Merrill Street'})))

[{u'_id': ObjectId('5647e57267df1b2b7ef62aab'),
  u'address': {u'housenumber': u'2039',
               u'postcode': u'95062',
               u'street': u'Merrill Street'},
  u'amenity': u'school',
  u'created': {u'changeset': u'11504010',
               u'timestamp': u'2012-05-05T03:58:04Z',
               u'uid': u'633791',
               u'user': u'sla29970',
               u'version': u'1'},
  u'id': u'1741019936',
  u'name': u'Cypress Charter High School',
  u'pos': [36.9650004, -121.9831159],
  u'type': u'node'}]
```

This time our assumption was correct. In the future we should be careful of streets of the same name but different type.

4.4 Amenities

```
In [39]: amenity = db.santacruz.aggregate([
                 {
                     '$match':{'amenity':{'$exists':1}}
                 },
                 {
                     '$group':{'_id':'$amenity', 'count':{'$sum':1}}
                 },
                 {
                     '$sort':{'count':-1}
                 },
                 {
                     '$limit':10
                 }
             ])

         print 'Top 10 Amenities'
         for doc in amenity:
             print doc

Top 10 Amenities
{u'count': 947, u'_id': u'parking'}
{u'count': 266, u'_id': u'bicycle_parking'}
{u'count': 249, u'_id': u'restaurant'}
{u'count': 221, u'_id': u'toilets'}
{u'count': 194, u'_id': u'bench'}
{u'count': 145, u'_id': u'place_of_worship'}
{u'count': 107, u'_id': u'school'}
{u'count': 99, u'_id': u'recycling'}
{u'count': 90, u'_id': u'cafe'}
{u'count': 67, u'_id': u'fast_food'}
```

It may seem surprising that the two most common amenity tags are parking and bicycle_parking. But our top contributor stevea is the the author of a local bicycle network numbering protocol, CycleNet.

4.5 Other

```
In [37]: cuisine = db.santacruz.aggregate([
                 {
                     '$match':{'cuisine':{'$exists':1}}
                 },
                 {
                     '$group':{'_id':'$cuisine', 'count':{'$sum':1}}
                 },
                 {
                     '$sort':{'count':-1}
                 },
                 {
                     '$limit':5
                 }
             ])


         religion = db.santacruz.aggregate([
                 {
                     '$match':{'religion':{'$exists':1}}
                 },
                 {
                     '$group':{'_id':'$religion', 'count':{'$sum':1}}
                 },
                 {
                     '$sort':{'count':-1}
                 }

             ])


         denomination = db.santacruz.aggregate([
                 {
                     '$match':{'religion':'christian','denomination':{'$exists':1}}
                 },
                 {
                     '$group':{'_id':'$denomination', 'count':{'$sum':1}}
                 },
                 {
                     '$sort':{'count':-1}
                 }
             ])


         print 'Most popular cuisine'
         for doc in cuisine:
             print doc

         print '\nReligions'
         for doc in religion:
             print doc

         print '\nChristian denominations'
```

```
        for doc in denomination:
            print doc

Most popular cuisine
{u'count': 44, u'_id': u'mexican'}
{u'count': 27, u'_id': u'pizza'}
{u'count': 24, u'_id': u'chinese'}
{u'count': 15, u'_id': u'burger'}
{u'count': 14, u'_id': u'japanese'}
{u'count': 14, u'_id': u'italian'}
{u'count': 11, u'_id': u'thai'}
{u'count': 11, u'_id': u'american'}
{u'count': 8, u'_id': u'regional'}
{u'count': 8, u'_id': u'sandwich'}

Religions
{u'count': 83, u'_id': u'christian'}
{u'count': 4, u'_id': u'buddhist'}
{u'count': 3, u'_id': u'jewish'}
{u'count': 1, u'_id': u'muslim'}
{u'count': 1, u'_id': u'pagan'}

Christian denominations
{u'count': 11, u'_id': u'baptist'}
{u'count': 8, u'_id': u'catholic'}
{u'count': 4, u'_id': u'presbyterian'}
{u'count': 4, u'_id': u'jehovahs_witness'}
{u'count': 3, u'_id': u'episcopal'}
{u'count': 3, u'_id': u'lutheran'}
{u'count': 3, u'_id': u'methodist'}
{u'count': 2, u'_id': u'evangelical'}
{u'count': 2, u'_id': u'anglican'}
{u'count': 2, u'_id': u'assemblies_of_god'}
{u'count': 2, u'_id': u'orthodox'}
{u'count': 2, u'_id': u'scientist'}
{u'count': 2, u'_id': u'mormon'}
{u'count': 1, u'_id': u'quaker'}
{u'count': 1, u'_id': u'seventh_day_adventist'}
{u'count': 1, u'_id': u'christ_scientist'}
```

4.6 Conclusion

The original Santa Cruz data was relatively clean. There were not that many over abbreviated streets and most zip codes were in proper 5-digit form. We saw that a fraction of a percent of docs have postal codes. We can sort of see why. In the top 10 amenities we have parking spots, bicycle parking, toilets, and benches. These amenities don't necessarily need specific addresses or postal codes. With that said, the amount of restaurants and places of worship outnumber the amout of documents with postal codes. Therefore, the data can be improved by adding complete addresses where needed.

Adding postal codes to addresses can be quite challenging. Not all documents include a city. We can use a different strategy, like basing postal codes on streets. But if we try to update postal codes based on street names, we run into problems when streets, such as Soquel Avenue, span multiple cities. With some knowledge of Santa Cruz County, we can write code for streets that we know span just one area. For example, if we know that Pacific Avenue has a beginning and end within the postal code 95060, then we can update the postal code for all documents with street Pacific Avenue.

We will always need to audit and clean our data, but the process can be made easier if we have accurate user input. I suggest that anyone interested in inputing or cleaning OSM data for Santa Cruz to get in touch with user stevea, as he is trying to develop practice for consistent input and refine tags that are used for specific amenities. It sounds like a fun project to take on as a hobby!