Udacity Data Analyst Nano Degree
Project 3 – Data Wrangling Submission
Ryan nelson

Does the project document the challenges encountered during the wrangling?

The actual structure of the data set is somewhat complicated. There are nodes that have individual tags and these are then assembled into other structures like ways and relations. From the structures it looks like there are multiple version of nodes. Since the tags are freeform and appended to nodes and ways it's ambiguous if there is a master node or way. I took the tack of ignoring nodes that didn't have Tag elements. The assumption being that if they don't have tag elements they are historical changes to the location and or are tracking version history.

To devel into the data a bit more I wrote a function that profiled the types of tags in the data

```
def profileData(filename):
  profile_file_name = 'profile_'+ filename
  with open(profile_file_name, 'w') as fp:
    tags = {}
    way_tags = {}
    tag_types={}

    csv_writer = csv.writer(fp)

    for event, elem in ET.iterparse(filename,  events=('start',)):
      try:
        tags[elem.tag] += 1

      except:
        tags[elem.tag] = 1
        print elem.tag

      for tag in elem.iter('tag'):
        try:
          way_tags[tag.attrib['k']] += 1

        except:
          print tag.attrib
          way_tags[tag.attrib['k']] = 1
      elem.clear()
      break

    fp.write(str(len(tags))+" diff tags in document:")
    for k, v in tags.items():
      csv_writer.writerow([k,v])

    fp.write('\ntag subtypes\n')
    for k, v in way_tags.items():
      csv_writer.writerow([k,v])
```

This produced a file with all of the tags in the file in them. It was a handy reference for tag data since I couldn't load the entire data set into memory and run queries against them.

Another point about the xml structure is since the tags aren't required we, by definition, can't assume that they will be there. Initially I was pulling out attributes like name and address tags, but it became apparent that this will throw exceptions I had one instance where a longitude tag wasn't found. So I wrapped the attribute extraction in it's own function that has a try except block. If the value isn't found it'll write the entire element to an error file that can be looked at later.

One of the more surprising things I encountered during this project is the effect of the data size on processing speed. I chose Orlando as my data set, and it was 128M. During my loading phase this took more than 5 min to extract tags and write it to a database, while that's not excessive that does seem to be a lot of time for such a small file. Once it was in the database though it was much faster to manipulate the data.

Is data cleaned programmatically?

This faster processing time helped a lot check for aberrant values. Using the following example code demonstrates the value of the database removing the need to do a lot of error handling with such unstructured data.

```
for record in  cur.find({'addr': {'$exists': True }}):
```

The first thing I checked was the street endings.

I used the following expected values dict:

```
expected_way_endings = ['Street', 'Avenue', 'Boulevard', 'Drive', 'Court',
'Play', 'Way', 'Pike', 'Parkway']
```

After processing I found the following strings with endings that weren't found:

```
[u'St.', u'Rd', u'390', u'136', u'Stars', u'North', u'Circle', u'250',
u'Highway', u'Row', u'Road', u'Trail', u'South', u'Hwy', u'9', u'Dr',
u'Loop', u'200', u'Lane', u'Plaza', u'F', u'303', u'St', u'260', u'Place',
u'5730', u'Broadway', u'Turnpike', u'103', u'100', u'104', u'Blvd.', u'1041',
u'6G', u'States', u'535', u'Blvd', u'Ave', u'434', u'blvd.', u'Ct']
```

Looking I can see that there are a few entries that should be added to my expected values, namely, ,
[u'Road', u'Trail', u'Turnpike', u'Loop']

After the dictionary was updated I wrote a method to programmatically change the string and update the database:

```
def fix_street_name(node):
  street_ending_re_search = street_ending_re.search(node['addr']).group()
    if street_ending_re_search in street_mappings:
      corrrected_addr = re.sub(street_ending_re, \
                street_mappings[street_ending_re_search], node['addr'])
      cur.update( {'_id':node['_id']}, \
                {'$set': {'addr':corrrected_addr }}, \
                upsert=False)
```

Despite the issues stated above the data was relatively clean. I ran checks against missing fields, including longitude, latitude, and username and user id. Since the fields are only written if they exist a simple existence check is enough to verify the data integrity. In addition we can reasonably assume some of this data should be paired, for example there should be both username and user id, and likewise longitude and latitude are paired.

```
def auditConsistency():
      query_missing_gps =  {'$or' :[{'long' :{'$exists': False}}, \
                                    {'lat':{'$exists': False}} ] }
      query_missing_name = {'name' : {'$exists': False}}
      query_missing_username =  {'username' : {'$exists': False}}
      query_missing_userid =  { 'userid' : {'$exists': False}}


      query_builder = {'type': 'node'}
      query_builder.update( query_missing_username)
      query_builder.update( query_missing_userid)

      for record in cur.find(query_builder):
            print record
```

Were I to keep playing with the data I'd probably look at making the data more succinct. The versioning information could probably truncated to reduce the overall OSM file size.

Another issues I have with the data is that a lot of the nodes seem to be revisions to the GPS coordinates. I'd like to see how much the coordinates deviate from each other and possibly do some sort of normalization on it. Or at least handle entries that are sufficiently different. A sum of squared computation can reduce the distance down to a scalar metric that can be used to converge the GPS coordinates. This would reduce the number of updates that aren't useful but would make it much harder to change dramatically different GPS information. It also preferences the first people to enter GPS locations and would need to remove outliers to improve the metric. All of this is in addition to the computing resources needed to crunch the metric for all of the locations.

Are overview statistics of the dataset computed?

Stats about data set:

```
      {'ways': 85145,
       'high_school_count': 11,
      'users': 679,
      'totalsize': 22249472,
      'nodes': 569564,
       'publix_count': 39,
       'elem_school_count': 41}
```

Code used to generate above stats:

```
def dbStats():
  stats = {
      'totalsize': db.command('collStats', DBNAME)['storageSize'],
      'users' : len(cur.distinct('userid')),
      'nodes' : cur.count( {'type':'node'} ),
      'ways' : cur.count( {'type':'way'} ),
      'publix_count' : cur.count({'name':re.compile('.*ublix.*')}),
      'high_school_count' : cur.count({'name': \
          re.compile('.*High School.*')}),
      'elem_school_count' : cur.count({'name': \
          re.compile('.*Elementary School.*')})

      }

      print stats
```