Written with StackEdit.

OpenStreetMap Data Case Study

Map Area

Las Vegas, Nevada, United States of America

- https://www.openstreetmap.org/relation/170117
- Metro Exracts Las Vegas
  10 MB bzip'ed XML OSM data

I was inspired to choose this map by hearing that my parents were about to take a trip to Las Vegas. I thought it would be an interesting city to look at.

Problems Encountered in the Map

One of the challenges in data wrangling is how to assess the quality of the data prior to transforming it into a database. First, I had to audit the data to discover values that might be invalid, inconsistent, incomplete, or not uniform. Values caught by the auditing could then be cleaned up, if necessary, in order to prepare a "tidy" (valid, consistent, complete, uniform) data set.

I looked for potential problems in my Las Vegas OSM file by running it against Python scripts based on the **Problem Set *"Case study: OpenStreetMap Data"* in Udacity's *Data Wrangling* course**, in particular the quiz files `tags.py`, `audit.py`, and `data.py`, which I had to complete in order for them to work.

I based my own code on these quiz scripts to audit, clean and transform the data in my OSM file. Here are some of the main problems I encountered in the data:

- Characters that would be invalid when transforming to CSV format.
- Invalid or non-uniform postal codes

- Unabbreviated state name
- Abbreviated street names
- Location-specific problems
- Too much data entered as a tag attribute's value, making the data appear invalid or inconsistent.
- "k" fields that make wrangling a more complicated chore

Problem Characters

Certain text characters were problematic in transforming to a plain text CSV-compatible format when using Python 2.7's `csv` module. Non-ASCII characters, for example, if left untouched could show up as the notorious blank character known as *"tofu"* (looks like an empty box). For example, diacritic characters (accents, umlauts, tildes), which are common in some non-English alphabets, might go missing.

```
<tag k="name" v="Café Berlin"/>
```

I wrote a script to clean problematic characters, `clean_chars.py`, imported it into a data transformation script, and then called functions from it in order to remove or replace problematic characters.

Within my `clean_chars.py` script, I included a function with a choice to encode text as UTF-8:

```
def process_values(tag, enc=False):
    if enc:
        val = tag.attrib['v'].encode('utf-8')  # Prevent encoding
errors
    else:
        val = tag.attrib['v']
    if has_problem(val):  # If val has problem characters:
        val = clean_problems(val)  # send val to get cleaned
    return val
```

From my data transformation script, I called this function with `enc` set to `True`:

```
for tag in element.iter('tag'):
    k = tag.attrib['k']
    v = clean_chars.process_values(tag, enc=True)
```

Postal Codes

I wanted the postal codes in my database to be strictly numeric, as they are in the U.S. However, some postal codes in the Las Vegas OSM file violated this uniformity. For example,

```
<tag k="addr:postcode" v="Nevada 89113"/>
```

Values such as this were discovered during the auditing phase of my wrangling. I adapted the `audit.py` script to find potential problems with numeric values in tags, and named the script `audit_numeric.py`. This script caught, for just one example, the "Nevada 89113" value, cited above.

I wrote an address cleaning script, "clean_address.py", similar to "clean_chars.py". Within this script I cleaned up problem postal codes this way,

```
def clean_postal(value):
    m = postal_re.search(value)
    if m:
        value = m.group()
    return value
```

where "postal_re" is a regular expression pattern:
`postal_re = re.compile(r'(\d{5})')`.

To keep the postal code field in the database uniform, I restricted it to five digits, omitting the extra four digits in any "ZIP + 4" style U.S. postal codes as well as omitting letters such as "Nevada" or "NV". I called this function from my my data transformation script, too.

Addresses

Several problems were encountered pertaining to street address data in the Las Vegas OSM map file.

Unabbreviated state name
I intended to replace the full state name "Nevada" with its two-letter U.S. abbreviation, "NV". However, Las Vegas is located in the far southeast corner of Nevada and the map could include points in surrounding states, so I included those state names as well. I mapped states to their abbreviations, then corrected where necessary:

```
states = {'Arizona': 'AZ', California': 'CA', 'Nevada': 'NV',
'Utah': 'UT'}

def clean_state(value):
    '''Ensure state value is a 2-character abbreviation'''
    if len(value) != 2 and value in states:
        value = states.get(value)
    return value
```

This was only ran against address tags in the OSM file, not all tags.

Abbreviated street names

I audited the Las Vegas OSM file for abbreviations in street names using the file `audit.py` provided as a quiz in **Lesson 6** of Udacity's *Data Wrangling course*. I altered this script slightly and saved it as `audit_address.py`. The results enabled me to create a mapping of street type abbreviations to full words that was location-specific to Las Vegas.

```
street_unabbrev = {
    'Ave': 'Avenue', 'AVE': 'Avenue', 'Ave.': 'Avenue',
    'Blvd': 'Boulevard', 'Blvd.': 'Boulevard',
    'Cir': 'Circle', 'Cir.': 'Circle',
    'Crt': 'Court', 'Crt.': 'Court', 'Ct': 'Court', 'Ct.': 'Court',
    'Dr': 'Drive', 'Dr.': 'Drive',
    'Fwy': 'Freeway', 'Fwy.': 'Freeway',
    'Hwy': 'Highway', 'Hwy.': 'Highway',
    'Ln': 'Lane', 'Ln.': 'Lane',
    'Mt': 'Mountain', 'Mt.': 'Mountain',
    'Pkwy': 'Parkway', 'Pkwy.': 'Parkway',
    'Pl': 'Place', 'Pl.': 'Place',
    'Pt': 'Point', 'Pt.': 'Point',
    'Rd': 'Road', 'Rd.': 'Road',
    'Rte': 'Route', 'Rte.': 'Route',
    'Sq': 'Square', 'Sq.': 'Square',
    'St': 'Street', 'St.': 'Street',
    'Ter': 'Terrace', 'Ter.': 'Terrace',
    'Tr': 'Trail', 'Tr.': 'Trail',
    'Wy': 'Way', 'Wy.': 'Way'}
```

I was pleasantly surprised at how few abbreviations were all UPPERCASE (only "AVE"), which spared me the trouble of having to write extra code to clean capitalization problems in street types. *Had I not audited the data first*, I might not have noticed this.

Abbreviated words occurred not only as street types (Rd. for Road, Ave. for Avenue, etc.) but also compass directions (e.g. N. for North, S. for South). I had to contend with both kinds of abbreviation in the map file.

In `clean_address.py` handling the direction words was straightforward:

```
nsew_re = re.compile(r'N |N\. |S |S\. |E |E\. |W |W\. ')

nsew_unabbrev = {'N ': 'North ', 'N. ': 'North ',
                 'S ': 'South ', 'S. ': 'South ',
                 'E ': 'East ', 'E. ': 'East ',
                 'W ': 'West ', 'W. ': 'West '}

def clean_nsew(name):
    '''In name, replace abbreviations with full directional
words'''
    m = nsew_re.search(name)
    if m:
        name = name.replace(m.group(), nsew_unabbrev[m.group()])
    return name
```

Handling abbreviated street types was mostly straightforward, too. Street names in Las Vegas typically have the street type at the very end (e.g., “West Tropicana Avenue”). Therefore, to find any abbreviations in those street names I was able to simply use the regular expression provided in the Udacity “audit.py” quiz script,

```
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
```

which locates the street type at the end of a street name string.

However, I discovered a notable, location-specific exception to that rule in Las Vegas, and it’s the world-famous “Las Vegas Strip”. It’s formal street name is actually either *“Las Vegas Blvd South”* or *“Las Vegas Blvd North”*. The street type is not at the end of the street name, which meant that the regex wouldn’t catch it. I handled this by adding a couple more lines of code to the `clean_streetname` function:

```
def clean_streetname(name):
    '''In name, replace abbreviated street types with full words'''
    m = street_type_re.search(name)
    if m and  m.group() in street_unabbrev:
        name = name.replace(m.group(), street_unabbrev[m.group()])
    # Munge the Las Vegas "Strip" because street-type not at end of
string
```

```
    if 'Vegas Blvd' in name:
        name = name.replace('Blvd', 'Boulevard')
    return name
```

After importing `clean_address` to the data transformation script, I called it like so,

```
if k.startswith('addr:'):
    tag_attrs['address'] = 'addressed'
    k = k.replace(':', '_')
    if k == 'addr_state':
        v = clean_address.clean_state(v)
    elif k == 'addr_postcode':
        v = clean_address.clean_postal(v)
    else:
        v = clean_address.clean_streetname(v)
        v = clean_address.clean_nsew(v)
    # set the key to cleaned value
    if k in fields:
        tag_attrs[k] = v
```

TIGER attributes
OSM way tags, but not node tags, contain TIGER (*Topologically Integrated Geographic Encoding and Referencing*) GIS data. Given the relative shortage of OSM-tagged address data (fields beginning with "addr:") in the map, it was useful to include TIGER data for wrangling addresses.

I found out how many tags use TIGER data in the process of examining the OSM file for all tag attributes. The following code captured and counted *all* the "k" attributes in way tags,

```
import xml.etree.cElementTree as ET
from collections import defaultdict
from sys import argv

script, in_file, out_file, elem_type = argv

tags = defaultdict(int)

for _, element in ET.iterparse(in_file):
    if element.tag == elem_type:
        for tag in element.iter('tag'):
            k = tag.attrib['k']
            #v = tag.attrib['v']
            tags[k] += 1

with open(out_file, 'w') as f:
    for k, v in sorted(tags.items(), key=lambda (k,v): v,
    reverse=True):
        f.write(k + ': ' + str(v) + '\n')
```

Way tags with TIGER data in “k” attribute, and their counts:

```
tiger:cfcc: 28553
tiger:county: 28693
tiger:mtfcc: 567
tiger:name_base: 27753
tiger:name_base_1: 2190
tiger:name_base_2: 98
tiger:name_base_3: 6
tiger:name_direction_prefix: 4060
tiger:name_direction_prefix_1: 281
tiger:name_direction_prefix_2: 9
tiger:name_direction_suffix: 17
tiger:name_direction_suffix_1: 10
tiger:name_direction_suffix_2: 1
tiger:name_full: 552
tiger:name_type: 26257
tiger:name_type_1: 1111
tiger:name_type_2: 44
tiger:name_type_3: 4
tiger:reviewed: 27102
tiger:separated: 20855
tiger:source: 21869
tiger:tlid: 21904
tiger:upload_uuid: 3302
tiger:zip_left: 24171
tiger:zip_left_1: 472
tiger:zip_left_2: 99
tiger:zip_left_3: 25
tiger:zip_left_4: 3
tiger:zip_right: 23858
tiger:zip_right_1: 276
tiger:zip_right_2: 63
tiger:zip_right_3: 12
tiger:zip_right_6: 2
```

Those “tiger:” attributes that are appended with underscores and digits appear to be redundant (and why was there a `tiger:zip_right_6` but not ones ending in “_4” or “_5”?). For example:

```
<way id="14321307"...
    <tag k="highway" v="residential"/>
    <tag k="name" v="Four Views Street"/>
    <tag k="tiger:cfcc" v="A41"/>
    <tag k="tiger:county" v="Clark, NV"/>
    <tag k="tiger:name_base" v="Four Views"/>
    <tag k="tiger:name_type" v="St"/>
    <tag k="tiger:reviewed" v="no"/>
    <tag k="tiger:zip_left" v="89143"/>
    <tag k="tiger:zip_left_1" v="89131"/>
    <tag k="tiger:zip_right" v="89143"/>
    <tag k="tiger:zip_right_6" v="89131"/>
```

and

```
<way id="14315941"...
    <tag k="highway" v="residential"/>
    <tag k="name" v="Avenida del Luna"/>
    <tag k="name_1" v="Avenida del Luna Avenue"/>
    <tag k="tiger:cfcc" v="A41"/>
    <tag k="tiger:county" v="Clark, NV"/>
    <tag k="tiger:name_base" v="Avenida del Luna"/>
    <tag k="tiger:name_base_1" v="Avenida del Luna"/>
    <tag k="tiger:name_type_1" v="Ave"/>
    <tag k="tiger:reviewed" v="no"/>
    <tag k="tiger:separated" v="no"/>
    <tag k="tiger:source" v="tiger_import_dch_v0.6_20070813"/>
    <tag k="tiger:tlid" v="201871430"/>
    <tag k="tiger:zip_left" v="89119"/>
    <tag k="tiger:zip_left_1" v="89119"/>
    <tag k="tiger:zip_left_2" v="89119"/>
    <tag k="tiger:zip_left_3" v="89119"/>
    <tag k="tiger:zip_left_4" v="89119"/>
    <tag k="tiger:zip_right" v="89119"/>
  </way>
```

For my purposes it was sufficient to transform to csv for the database using just the fields “tiger_zip_left” and “tiger_zip_right”.

Another point to note about the enumeration of “k”s in tag elements and their attribute counts (*all of them, not just TIGER tags, in both ways and nodes*) is the *sheer number* of unique tag elements. In the Las Vegas OSM file that I downloaded and parsed, many of these tags’ “k” attributes appear only once.

- Out of *267* unique “*k*“s in node tags, 59 of them appear *only one time*, and just under half of them appear 10 or fewer times. (In a document containing over *800,000* node elements.)

- Out of *183* unique “*k*“s in way tags, 45 of them appear *only one time*, and more than half of them appear 10 or fewer times. (In a document containing over *72,000* way elements.)

I learned in this data wrangling exercise that you can wind up with a lot of superfluous or trivial data to fill a database with. It would probably take more comprehensive data analysis to decide how much of it (if any) is worth including in a data set.

Inconsistent, Invalid data entries

After importing the Las Vegas data into a **SQLite3** database, I began exploring features of interest. It was here that I began noticing other data inconsistencies and nonconformity.

Example 1

```
SELECT DISTINCT natural, waterway, water, name
FROM way_tags
WHERE natural like '%w%';
```

returned a table whose very first row was:

```
natural  waterway  water  name
-------  --------  -----  ----
water                     Fountains of Bellagio
```

Note that the key is “natural” – but it’s a pretty safe guess that “Fountains of Bellagio” is not a naturally-occurring water feature!

Example 2
Full address in one field. Addresses not split up into more specific fields. I discovered this in the course of doing something unrelated: I was searching the database for problematic postal codes (discussed earlier).

```
SELECT tourism, name
FROM (SELECT tourism, name FROM node_tags
UNION SELECT tourism, name FROM way_tags)
GROUP BY name
HAVING name GLOB '*[0-9][0-9][0-9][0-9][0-9]*';
```

```
hotel|Circus Circus 2880 Las Vegas Boulevard NV 89109 Las Vegas
Vereinigte Staaten von Amerika
hotel|Element Las Vegas Summerline 10555 Discovery Drive
```

As you can see, for these two hotel names, the *entire address* was entered.

Data Overview

File sizes (rounded to nearest MB)

```
lasvegas.osm      151 MB
lasvegas.db       127 MB
LV_node_tags       42 MB
LV_nodes.csv       57 MB
LV_way_nd.csv       8 MB
LV_way_tags.csv    10 MB
LV_ways.csv         4 MB
```

Number of OSM nodes and ways

```
SELECT COUNT(*) FROM nodes;
```

681,504 *nodes*

```
SELECT COUNT(*) FROM ways;
```

72,817 *ways*

OSM User Data

Number of unique users

```
SELECT COUNT(DISTINCT user)
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways);
```

440 *users (contributors)*

Top 10 contributors to map, by number of elements entered

```
SELECT DISTINCT user, COUNT(*) AS count
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways)
GROUP BY user
ORDER BY count DESC LIMIT 10;
```

```
alimamo         255374
woodpeck_fixbot  83303
nmixter          70222
gMitchellD       52693
robgeb           45572
MojaveNC         34761
nm7s9            17397
balrog-kun       16153
ocotillo         13256
TIGERcnl         12310
```

Other Data Queries, and Suggestions for Improvements

Top 20 Amenities

```
SELECT amenity, COUNT(*) AS count
FROM(SELECT amenity FROM node_tags WHERE amenity != ''
UNION ALL
SELECT amenity FROM way_tags WHERE amenity != '')
GROUP BY amenity
ORDER BY count DESC LIMIT 20;
```

```
amenity                    count
-------------------------  ----------
parking                    586
```

```
school                      526
place_of_worship            363
fountain                    265
restaurant                  126
fast_food                   88
fire_station                68
hospital                    68
fuel                        61
post_office                 59
shelter                     50
toilets                     41
public_building             40
bar                         38
cafe                        35
bank                        24
casino                      24
theatre                     24
library                     22
swimming_pool               20
```

Only 24 casinos in Las Vegas? That didn't look like an accurate reflection of reality. But then of course my query only captured nodes and ways *actually tagged with* "amenity" equal to "casino".
I suspected that there were more than that, so I expanded the query.

```
SELECT DISTINCT amenity, tourism, name FROM node_tags
WHERE name LIKE '%casino%' OR amenity LIKE '%casino%'
UNION ALL
SELECT DISTINCT amenity, tourism, name FROM way_tags
WHERE name LIKE '%casino%' OR amenity LIKE '%casino%'
ORDER BY name;
```

```
amenity     tourism     name
----------  ----------  ----------------------------------------
casino
casino      hotel       Aria Resort & Casino
casino      hotel       Bally's Hotel and Casino
casino      hotel       Bellagio Hotel and Casino
casino                  Bill's Gamblin' Hall & Saloon
            hotel       Boulder Station Hotel and Casino
bar                     Cabana Bar & Casino
casino      hotel       Caesars Hotel and Casino
parking                 Casino Parking Garage
casino      hotel       Circus Circus Las Vegas Hotel and Casino
casino                  Club Fortune Casino
```

```
            hotel       Encore Casino at Wynn
                        Encore Hotel & Casino
casino      hotel       Excalibur Hotel and Casino
casino      hotel       Gold Coast Hotel & Casino
            hotel       Hacienda Hotel and Casino
casino      hotel       Hard Rock Hotel and Casino
casino      hotel       Imperial Palace Hotel and Casino
            hotel       LVH Las Vegas Hotel & Casino
restaurant              Lake Mead Casino
casino      hotel       Luxor Hotel and Casino
            attraction  M Resort & Casino
casino      hotel       MGM Grand Hotel and Casino
parking                 Main Street Casino Parking
            hotel       Main Street Station Casino and Hotel
            hotel       Mandalay Bay Hotel and Casino
casino      hotel       Mandalay Bay Hotel and Casino
casino      hotel       Mirage Hotel and Casino
casino      hotel       Monte Carlo Hotel and Casino
casino      hotel       New York New York Hotel and Casino
                        North Casino Center Boulevard
                        Orleans Casino
casino      hotel       Palms Casino Resort
casino      hotel       Planet Hollywood Hotel and Casino
                        Railroad Pass Casino
casino                  Rampart Casino
                        Silverton Hotel and Casino
                        South Casino Center Boulevard
casino      hotel       Stratosphere Hotel and Casino
            hotel       Suncoast Hotel and Casino
                        The California Hotel and Casino
casino      hotel       Treasure Island Hotel and Casino
casino      hotel       Venetian Hotel and Casino
            hotel       Wynn Hotel & Casino
```

So, there are Hotel/Casinos tagged as “casino” but not “hotel”, or tagged as “hotel” but not “casino”, and a few not tagged as either. A suggestion for improvement here would be to bring this to the attention of the establishments themselves, to improve the establishment’s visibility in OSM searches. A freelancer could offer to properly and cleanly tag the attributes in OpenStreetMap for an establishment, but it’s hard for me to tell from OSM’s web site whether this would violate OSM acceptable use policies.

Addresses

States in addresses

```
SELECT addr_state, COUNT(*) AS count FROM
(SELECT addr_state FROM node_tags WHERE address = 'addressed'
UNION ALL SELECT addr_state FROM way_tags WHERE address =
'addressed')
GROUP BY addr_state
ORDER BY count DESC;
```

```
addr_state    count
------------  ------
NV            519
AZ            39
CA            1
```

This state query is based on data that was cleaned prior to importing to database. Had I not cleaned the data, the “addr_state” field would’ve contained spelled out state names (“Nevada”). I think an improvement could be made for countries that use postal abbreviations (such as the two-letter codes used by the U.S. and Canada) by putting in a constraint on what can be entered in an OSM map.

Some names of places from the “place” field:

```
SELECT * FROM
(SELECT place, name FROM node_tags
WHERE place IN ('city', 'town', 'village')
UNION
SELECT place, name FROM way_tags)
WHERE place IN ('city', 'town', 'village')
ORDER BY place;
```

```
place            name
---------------  --------------------
city             Henderson
city             Las Vegas
city             North Las Vegas
town             Boulder City
town             Enterprise
town             Jean
town             Moapa Valley
town             Summerlin
town             Whitney
town             Winchester
village          Blue Diamond
village         Corn Creek
village         Goodsprings
village         Logandale
village         Mountain Springs
village         Overton
village         Temple Bar
```

The city portion of the address of an element could be gotten from fields other than those that begin with “addr:”, “tiger:” or “gnis:”. This would require more data cleaning and transformation code that is beyond the scope of my project, but is food for thought.

Other ideas about the datasets, and Suggestions for improvement

The open source nature of OpenStreetMap leads in some cases to inconsistent data and a lack of standardization.

The example, above, where *Circus Circus’s* entire address was entered into the field, suggests that the source of the data error was a user without a proper understanding of how to tag the location in OSM. (By the way, *“Vereinigte Staaten von Amerika”*, is German for “United States of America”; it’s not unrealistic to suppose the user was a tourist.)

Sometimes, the preferred format of a tag might be unclear to a user of a map.
For example, a couple of tags had me wondering whether the field should be Boolean values or text.

A query of the “building” field resulted in both *yes* | *no* values and more descriptive *text* values:

```
SELECT building, COUNT(*) AS count
FROM (SELECT building FROM node_tags
UNION ALL SELECT building FROM way_tags)
GROUP BY building
HAVING building != ''
ORDER BY count DESC LIMIT 15;
```

```
building        count
--------------- ------
```

```
yes                 2364
hangar              175
warehouse           134
roof                70
retail              69
house               53
industrial          43
office              41
entrance            33
residential         16
apartments          12
garage              12
public              3
hut                 2
no                  2
```

Apparently, this is *not a bug but a feature*. The OSM wiki states

> **Buildings can simply be `building=yes` or use a value that describes the building typology, for example `building=house, building=hut, building=garage, building=school`.**

Unless you're a contributor to a map and have bothered to read the wiki guidelines, this might be confusing or counterintuitive.

OSM declares that "OpenStreetMap emphasizes local knowledge." But, at the risk of nit-picking, I think the ***tagging of sources involving "local knowledge" could use cleaning up:***

For example, I came across 6 distinctive formats in which the existence of local knowledge has been tagged in the OSM map.

```
SELECT source, COUNT(*) AS count
FROM (SELECT source FROM node_tags
UNION ALL SELECT source FROM way_tags)
GROUP BY source
HAVING source LIKE '%know%';
```

```
source                            count
--------------------------------  ----------
local knowledge;Yahoo image        14
Yahoo image; local knowledge       11
local knowledge                    10
Local Knowledge                    7
```

```
knowledge                        1
personal knowledge               10
```

Redundancy in feature key words and tag values

```
SELECT *, COUNT(*) AS count FROM
(SELECT highway, NULL 'foot', NULL 'footway' FROM node_tags
UNION SELECT highway, foot, footway FROM way_tags)
GROUP BY highway
HAVING highway IN ('footway', 'crossing', 'path', 'steps',
'pedestrian');
-- note: _NULL 'foot', NULL 'footway'_ allows the UNION operation
to work by equalizing the number of columns in the tables being
UNION-ed.
```

```
highway     foot  footway    count
----------  ----  ---------  -----
crossing    NULL  NULL       1
footway     yes   sidewalk   10
path        yes   sidewalk   7
pedestrian  yes              2
steps       yes              3
```

I found that “footway” is both a value assigned to the “highway” key *and* a key in its own right. This seemed redundant to me. So does the presence of a key-value pair of `"foot"="yes"` in a tag that already includes a “footway” key, since by definition a footway is a path take on foot. I also noticed that “footway”=”sidewalk” not only when `"highway"="footway"`, *but also when* `"highway"="path"`. So, maybe there’s some room for improvement here, perhaps by not allowing the “highway” key to be set to “footway”, for starters.

Additional Data Exploration

The **FIXME** key.
OpenStreetMap’s “fixme” key is used for human-entered, not automated, map data, in order “to express that the mapper thinks there is an error” and needs further attention.

```
SELECT FIXME, type, COUNT(*) AS count
FROM (SELECT FIXME, type FROM node_tags
UNION ALL SELECT FIXME, type FROM way_tags)
```

```
GROUP BY FIXME
HAVING FIXME != '';
```

```
FIXME                                                         type
count
------------------------------------------------------------  -----
-  ------
access=private?                                               way
1
are bikes allowed?                                            way
148
Area Needs Checking                                           way
1
check lanes                                                   way
50
check lanes; are bikes allowed?                               way
11
check name                                                    way
1
Continue                                                      node
2
Divided highway                                               way
45
Divided highway.                                              way
8
Does the old_ref apply to the entire length?                  way
7
dual carriageway                                              way
14
inaccurate                                                    way
1
Is this really a park & sports center?                        way
1
Landuse shouldnt be attached to centerlines but follow actu   way
1
name needs checking                                           way
1
need subdivision                                              way
1
not sure which shop type                                      way
1
Old alignment being replaced                                  way
3
old_ref?                                                      node
1
reconfigured?                                                 way
6
```

```
ref?                                                          node
1
Remove this road when adjacent ramp is completed.             way
1
Temporary construction road.                                  way
4
This way will go oneway when other side is completed.         way
2
verfiy bicycle=yes                                            way
103
verify bicycle=yes                                            way
25
verify entrance type                                          node
4
yes                                                           way
2
```

Questions about bicycle usage predominate this field.
4 of the top 7 counts are bicycle-related.

```
SELECT SUM(FIXME LIKE '%bi%')
FROM (SELECT FIXME FROM node_tags
UNION ALL SELECT FIXME FROM way_tags);

287

SELECT SUM(FIXME != '')
FROM (SELECT FIXME FROM node_tags
UNION ALL SELECT FIXME FROM way_tags);

446
```

And 287 out of 446 “FIXME” rows (64%) are bicycle-related.

Conclusions

My overall impression of the Las Vegas OpenStreetMap data is that it is a work in progress, as befits an open source effort. Contributors have mostly made a good effort to stick with OSM conventions, but there is plenty of room for increased clarity – More standardization of tags, elimination of underused or misused tags, and tighter control of data quality (a sort of selfregulating of contributions, such as the sort we see on Wikipedia). Furthermore, I think there is a wide, untapped opportunity for businesses (such as the casino/hotels mentioned earlier) to fill in data about themselves in OSM maps. It leaves me wondering how much awareness about OpenStreetMap there actually is. Perhaps the OSM community could do more to promote itself as an alternative to Google and Bing Maps.

Nevertheless, for a project of its kind, OSM has done well to serve its purposes.

The data wrangling, auditing, cleaning, and transformation steps of this project were more problematic than the querying and overview of the database that resulted from it.

Appendix

(***A note on Udacity:***
At the time of my data wrangling, cleaning, and transformation, I was not yet officially enrolled in the Udacity Data Analyst Nanodegree, just doing it “for free”. Because I wasn’t enrolled, the *“Preparing For Database - SQL”* lessons weren’t visible to me. So, in writing my data transformation code as well as a sql schema, I had to wing it. I based my Python data transformation code on the code intended for prepping the *“MongoDB Case Study”*, which was visible to me, and which I adapted to write flat CSV that could be imported to a SQLite database.)