1 OpenStreetMap Data Project - SQL

1.1 Map Description

The greater Phoenix, Arizona area.

As I will be moving to Scottsdale, Arizona, I chose to investigate the OpenStreetMap data for the greater Phoenix Metro Area which includes the suburban regions of Scottsdale, Apache Junction, Fountain Hills, Glendale, Tempe, Mesa and several others including the city of Phoenix itself.

Data was retrieved from the `http://overpass-api.de/query_form.html` query form using a latitude, longitude bounding box of the Phoenix area and data saved to the default osm format. The greater Phoenix area data file sizes are shown:

```
artdegraw ~/Proj3_Py27_SpyProj > ls -l Phx_metro*
-rw-r--r-- 1 artdegraw artdegraw 320561152 Feb  9 23:47 Phx_metro.db
-rw-r--r-- 1 artdegraw artdegraw 580518138 Feb  7 09:08 Phx_metro.osm
```

1.2 Problems in the Map Data

There were several problems that I noticed in the Phoenix metro data. These problems are listed below:

1. Inconsistent 'postcode' representation: Some zip codes were represented as 5 digit and some as 9 digit. Others contained the leading string 'AZ '.

2. Inconsistent county representation: In the vast majority of cases the county was listed as 'Maricopa' but in some it was given as 'Maricopa AZ'

3. Inconsistent city names: these include case issues such as 'MEsa' vs 'Mesa' or 'Laveen' vs 'Laveen Valley' or just simple misspellings.

4. Incorrect node location: There is at least one node that appears to be a zip code that is not compatible with southern Arizona.

As the Phoenix metro OpenStreetMap data set was rather large, a smaller subset consisting of the Scottsdale area was first used as test set for the programmatic features that manipulate the data. The first two problems mentioned above could be treated rather simply with a string manipulation. The data was extracted from the osm file and into separate csv files using a modified **data.py** file (see section 4.1 for code). To construct the sqlite3 database with tables created using the provided schema, a function was made to both create the tables and load them from the csv files (see section 4.2 for code).

1.3 Creating Consistent Zip Codes

The inconsistency in zip code representation came in three varieties: 5 digit code, 9 digit code with 5 digit hyphen 4 digit, and a five digit with a leading 'AZ ' string as in the examples '85302', '85260-5518', 'AZ 85007'. On the smaller Scottsdale data set these three were the only issues with zip codes. On the larger Phoenix metro data set however, there were others such as single digit and some nine-digit proceeded by 'AZ ' and some with special characters (specified by '\xe2 \x80 \x8e'). The sequence of scripts will address most of these concerns.

- To address the zip codes that started out with 'AZ ', the relevant .csv files were loaded into a Pandas data frame directly from the .csv files. The 'AZ ' was then stripped from the zip codes that contained it. To address the 9 digit zip vs. the 5 digit zip I ran a count of the lengths of the zip codes which showed an overwhelming majority have only a 5 digit zip code. So for simplicity I converted all 9 digit to 5 digit by simply splitting the string at '-' and keeping the value of the first 5 digits as the new zip code. To check the results, once the database was loaded with the auditted values:

```
sqlite> select value, count(*)
   ...> from nodes_tags
   ...> where nodes_tags.key = 'postcode'
   ...> group by nodes_tags.value
   ...> order by count(*) desc;
```

zip	count
85028	1701
85301	652
⋮	⋮
92127	1
92509	1
95295	1

As noted previously, there are zip codes that appear to belong to Southern California: 92127 - San Diego, 92509 - Riverside, and an invalid one 95295 - invalid zip code according to usps.com.

1.4 City Name Inconsistency

The city names seem to have some misspellings and capitalization issues.

```
sqlite> select tags.value, count(*) as count
   ...> from (select * from nodes_tags union all
   ...> select * from way_tags) tags
   ...> where tags.key = 'city'
   ...> group by tags.value
   ...> order by count desc;
```

City	count
Phoenix	20844
Glendale	4872
Scottsdale	1745
Mesa	1137
Peoria	1001
⋮	⋮
Tohono Oodham	1
Tollenson	1
mesa	1
peoria	1
scottsdale	1
sun City West	1
tEMPE	1

To help remedy the city name inconsistencies, the names were compared to a city/town list for Maricopa County, Arizona, obtained from `http://phoenix.about.com/od/govtcity/qt/cities-towns-maricopa-county.htm`. City names were compared using the **SequenceMatcher** method from the **difflib** package. This feature determines a numerical score for similarity between two strings. If the score is near 1.0 then the strings are considered to be quite similar. If the score is near 0.0 then the strings are considered to be rather dissimilar. The strings were compared in a way that did not account for case sensitivity to help pick up on misspellings in the score and not capitalization issues. That is, two strings with the same spelling with varying capitalization would score a 1.0, showing that they are infact the same word with two different representations. The function that was written can address both case sensitivity and insensitivity in the scoring

While this did not fix the entire problem, it at least helped to resolve it. For analysis purposes, I created a list of unmatched city names, that is names that did not meet the criteria as having a city/town name in the list that was 'close enough' to any element of the list . For the **ways_tags** data the unmatched names were

```
{'2036 N. Gilbert Rd.', 'Gold Canyon', 'Laveen', 'Laveen Village', 'Maricopa', 'Riverside',
 'San Diego', 'San Tan Valley', 'Sun City', 'Sun City West', 'Tohono Oodham', 'Wittmann'}
```

The list of unmatched names is rather short as hoped. This list does contain some southern California cities as suspected. The apparent street address entry of '2036 N. Gilbert Rd.' was edited to now have a 'key' value of 'street' instead of 'city' as it was here.

It should be noted that there was some similarity among the city names themselves. The names of "Cave Creek" and "Queen Creek" at 2/3 had the highest similarity score. This could lead to a misspelling of one being mistaken for a misspelling of the other. However, it should be the case that a misspelling of "Cave Creek" should still yield a higher similarity score than it would to "Queen Creek", as the 'best_match' function searches for the match with the highest score not just a large one. But this is a concern, nonetheless. To help fix this concern, the threshold score of 0.6 could be increased, but this would also diminish the ability of the task to catch the names that were spelled rather poorly.

1.5 City Versus Zip

To partially address the potential mislabeling of addresses within the valley as simply 'Phoenix' I performed a sql query on the cities and postcodes. Since the data that I wanted to related lived in the same table (key=city and key=postcode) I performed a self join. What I was looking for was whether or not one postcode had associated with it more than one city. This would stand out as a potential error since postcodes are a refinement of the city designation for postal addresses. The code and result:

```
select t.id, t.value, s.value
from(
        select id, value
        from nodes_tags
        where key = 'city') t
        join
        (
        select id, value
        from nodes_tags
        where key = 'postcode'
        ) s
where t.id = s.id
order by s.value;
```

id	city	postcode
359293112	Phoenix	85003
2469618342	Phoenix	85004
3833145142	Phoenix	85006
4560431809	Phoenix	85007
⋮	⋮	⋮
2074055545	Wittmann	85361
4329328824	Surprise	85374
4422428383	Sun City West	85375
3085412764	Peoria	85381
3224573738	Peoria	85382

1.6 Street Abbreviations

For the 'Street' vs. 'St' vs. 'St.' inconsistencies and the like I used a straight string comparison in a list with punctuation removed only on the 'key' = 'street' elements. The decided convention was to use eliminate the abbreviations and include the entire word. As there are several common abbreviation for different words, the potential abbreviations were loading into a list and compared to words in the street name after stripping away punctuation. For ease of application, a function was written to also accept a data frame and run the code over the appropriate column in the data frame. The code was run on the **ways_tags** and **nodes_tags** Pandas data frames.

```
import string
conventions = [['West','W'],['North','N'],["South","S"],['East','E'],\
                ['Street',"St"],["Road","Rd"],["Avenue","Av","Ave"],\
                ["Place","Pl"],["Boulevard","Blvd"],["Trail","Tr"],\
                ["Place","Pl"],["Highway","Hwy","Hw","Hy"],["Parkway","Pkwy","Pw"]]

num_conventions = len(conventions)

def conv_street(street_name):
    split_up = street_name.split(" ")
    for i in range(len(split_up)):
        for j in range(num_conventions):
            #Strip punctuation and compare to naming conventions list
            if split_up[i].translate(None, string.punctuation) in conventions[j]:
                split_up[i] = conventions[j][0]
    whole = ' '.join(c for c in split_up)
    return whole

def conv_df(df):
    for i in range(len(df)):
        if df.loc[i,'key'] == 'street':
            df.loc[i,'value'] = conv_street(df.loc[i,'value'])
    return df
```

1.7 State Values

A similar procedure was applied to the 'key' = 'state' field as the variation in how the state was identified was large: {'A', 'AS', 'AZ', 'AZ (Arizona)', 'AZZ', 'Arizona', 'Az', 'TX', 'US', 'az'}

Any value contained in {'A', 'AS', 'AZ', 'AZ (Arizona)', 'AZZ', 'Arizona', 'Az', 'az'} was set to simply 'AZ' as long as the field 'type' was 'addr'. There was another value for the 'type' field of 'is_in' which was specifically left alone. I only felt it necessary to change the 'addr' type values since the postal address abbreviation is 'AZ'

2 Some SQL Database Queries

The number of unique users in the ways and nodes tables is 1216 and a simpler query yields the number of ways and nodes as 334539 and 2383267 respectively as shown below.

```
sqlite> select count(distinct user) from
   ...> (select user from nodes union select user from ways) tags;
1216
sqlite> select count(*) as c from nodes;
2383267
sqlite> select count(*) as c from ways;
334539
```

Keeping the ways and nodes tables separate the number of designated coffee or cafe points in each table is 74 and 316 respectively.

```
   ...> way_tags
   ...> where value like '%cafe%'
   ...> or
   ...> value like '%coffee%';
74
sqlite> select count(*) as c
   ...> from nodes_tags
   ...> where value like '%cafe%'
   ...> or
   ...> value like '%coffee%';
316
```

Some slightly more complicated queries are contained in the subsection "City Versus Zip" and the following section.

3 Further Work

It would be interesting to see how the nodes are distributed around the suburban areas. Are they essentially uniformly scattered or are they concentrated in certain areas? If the data is user contributed, my guess is that they would tend to be concentrated around places that are most visited rather than uniformly distributed. To test this theory, one could look at the distribution of nodes as an overlay on a map of the Phoenix area. The difficulty here is not in the data retrieval, as this is all contained in the nodes tables but in the display of the data. Some information on using Python to display maps and data can be found at `http://sensitivecities.com/so-youd-like-to-make-a-map-using-python-EN.html#.WKKF0lUrJpg` But, since this is a project on using map data and not creating a map, I instead thought to that one could simply average the latitude / longitude values for each suburban town/city and see how close this was to the actual city center or commercial center of the town/city. An example for Scottsdale, AZ is shown below using SQL:

```
select avg(nodes.lon), avg(nodes.lat)
from nodes join nodes_tags
on nodes.id = nodes_tags.id
where nodes_tags.key = 'city'
and nodes_tags.value = 'Scottsdale';
-111.954555026667|33.5950626666667
```

If I had to pick a "city center" of Scottsdale, I would choose the historic or Old Town Scottsdale which is located around the geographic coordinates of -111.91126 longitude and 33.494107 latitude. The SQL results are a point that is nearly 10 miles away but is more in the geographic center of Scottsdale. This may make one to lean towards the points in the nodes data set to be more uniformly spread about the city, but it is only one of many suburbs.