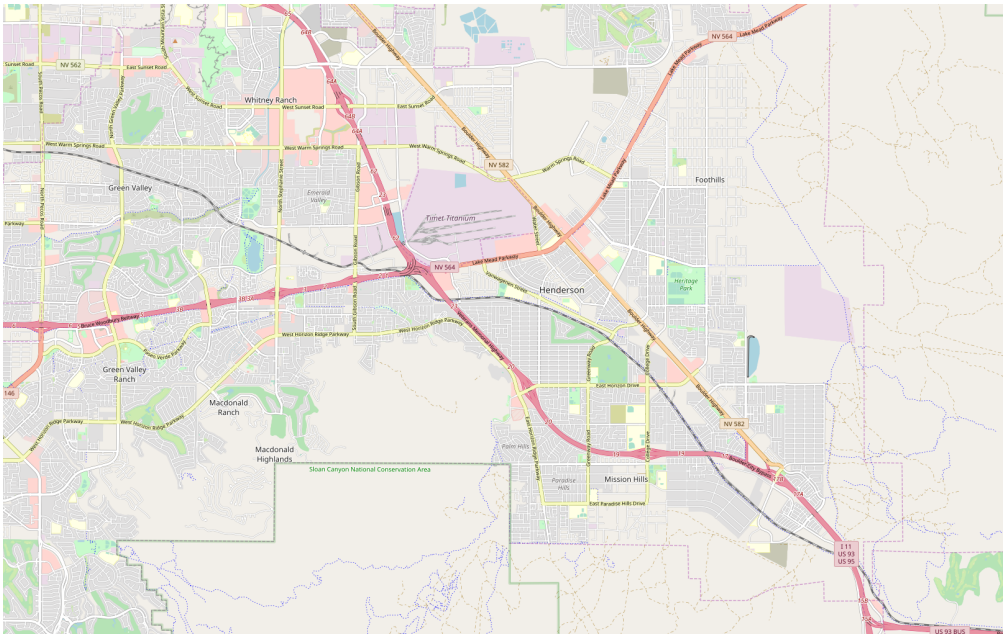
Wrangling OpenStreetMap Data

June 2, 2019

1 Wrangling Open Street Map Data

1.0.1 Introduction

This project aims to analyze a subset of the Open Street Map ("OSM") data for my hometown of Henderson, Nevada. Data is retrieved from OSM through a third-party API called overpass in a XML based .OSM file format. A data auditing and cleaning process is applied to the data and a new "cleaned" dataset is output in an XML file format. The data is then tranformed into more "data science friendly" Pandas DataFrames for analysis and future modeling. The DataFrames are exported to .CSV file format where they are then imported into a SQLite database for querying and analysis by others..



Henderson,NV

Map Area - Henderson, Nevada, United States

1.0.2 Challenges Encountered

Data Cleanliness

- The vast majority of address fields located in this dataset were correct. However there were a few that needed correction such as "Dr." for drive and "St" for street. These were easily cleaned with a function and regular expressions. However, there were some records that were not rapidly apparent as to their nature (i.e. "Avenue I"). Internet searches were unfruitful so these values were stripped of their trailing letter and left as "Avenue".
- All Latitude/Longitude/ZipCode/County fields were correct at the time of this writing. More time was spent on ensuring functions were created to catch bad data that I did not consider this data would be very clean already.

Data Structure

- While universal, XML made the task of retrieving data a bit more complicated than it would have been provided in a JSON or tabular file format.
- Instead of relying on attributes, the data instead opted for an almost pure Key->Value relationship. This required extra attention and processing of data in parts of the data utilizing predominantly Key-Value (tag data).
- Due to the vertical structure of the data (instead of a more traditional tabular structure), all values from keys had to be stored in the database as a text data type. There were several candidates that could have used a space saving data type such as int or float. With a schema redesign and some pivoting of the source DataFrames, this issue could be resolved.

Variety of User Types Entering Data

- From analyzing and inspecting the data, it would appear that users editing the OSM data were likely doing so for a variety of reasons. In my particular region, there was not a lot of "fun" (restaurants, etc.) data compared to more industrial data such as locations of power lines/highway trails/etc. This is discussed further in the Section **??** section
-

1.1 TIGER entered data often duplicated fields that already existed in the dataset. If TIGER is to be the authority of certain fields in the data, it may be a consideration to replace user entered fields with those entered by TIGER

1.1.1 Data Processing/Auditing/Cleaning

Audit.py - Data is read in from the Henderson.osm xml file - Data is iterated through and the following columns are cleaned/verified: street, county, zipcode, latitude, longitude - Records falling outsides norms are printed to the user - Once records have been iterated and cleaned, the cleaned data is output to the Cleaned_Henderson.osm XML file

Convert_XML_to_DataFrame_CSV.py - Cleaned data is iterated through and converted to a Pandas DataFrame for easy analysis/operations - Data is analyzed for columns of interest or abnormalities - Some additional cleaning is done (replacing Yes/No values with True/False strings) -

Each DataFrame represents a SQL table that will be created. For each DataFrame the data types are enforced and any missing values are given an empty string. Finally columns are put into the proper order the SQL database expects. - Each DataFrame is exported to CSV file format for easy import into SQL tables > **Analyst Note**: Jupyter Notebook avaliable with notes detailing steps taken/analysis completed

create_SQL_tables.py - After a database has been created in the SQLite shell, a connection to the DB is made via Python - Queries are created for each table to be created - The queries are iterated through and executed, creating the tables in our database - SQLite shell commands are run importing our data from the .CSV into our newly created tables > **Analyst Note**: Jupyter Notebook avaliable with steps and notes as well as SQLite shell commands —

1.1.2 Dataset Statistics/Items of Interest

Function to Retrieve Custom SQL Queries and Output as DataFrame Object This function allows analyst to easily write custom SQL queries and send them directly to our SQLite server. Once retrieved, records are output/displayed as a Pandas DataFrame for easy maniupulation/plotting/and displaying in Jupyter Environment.

[4]:
```
import sqlite3 as sq
import os
import pandas as pd
import re

filepath = r'S:\Code\School\WGU_DataAnalyst_NanoDegree\02 - Data Wrangling with␣
↪MongoDB'
```

Function:

[2]:
```
def return_sql_query_as_dataframe(sql_query):
    #Connect to SQL DB
    connection = sq.connect(os.path.join(filepath, 'MapData.db'))
    cursor = connection.cursor()

    #Send user query to Cursor
    query = sql_query
    cursor.execute(query)

    #Fetch results and return a Pandas DF
    result = cursor.fetchall()
    names = [description[0] for description in cursor.description]
    return(pd.DataFrame(result, columns=names))
```

Files and File Sizes List files sizes for .OSM XML files and output .CSV files

[5]:
```
for root, dirs, files in os.walk(filepath):
    for fn in files:
        if re.search('\.[csv|.osm]', fn):
            path = os.path.join(root, fn)
            size = os.stat(path).st_size # in bytes
            print(fn,"--", "{}KB".format(str(round(float(size/1024), 1))))
```

```
Cleaned_Henderson.osm -- 76697.4KB
henderson.osm -- 75833.1KB
nodes.csv -- 31376.1KB
nodes_tags.csv -- 557.8KB
ways.csv -- 2313.8KB
ways_nodes.csv -- 10428.6KB
ways_tags.csv -- 3538.6KB
```

Number of Unique Users

[6]:
```
query = '''select count(distinct b.user) as "Distinct Users in Both Tables"
from (
select user from nodes
union
select user from ways)b;'''
return_sql_query_as_dataframe(query)
```

[6]:
```
   Distinct Users in Both Tables
0                            394
```

Number of Nodes in Dataset

[7]:
```
return_sql_query_as_dataframe('select count(*) as "Number of Nodes in Dataset"␣
 ↪from nodes;')
```

[7]:
```
   Number of Nodes in Dataset
0                      353779
```

Number of Ways in Dataset

[8]:
```
return_sql_query_as_dataframe('select count(*) as "Number of Ways in Dataset"␣
 ↪from ways;')
```

[8]:
```
   Number of Ways in Dataset
0                      35264
```

Retrieve Number of Chosen Type of Node (Power Towers) Look at sample head of DataFrame returned containing power towers

[9]:
```
query = '''select * from nodes n
inner join nodes_tags nt on nt.id = n.id
where nt.key = 'power'
limit 10000;'''
return_sql_query_as_dataframe(query).head()
```

[9]:
```
         id        lat         lon                     user     uid  version  \
0  54315452  35.311169 -116.247819  Chris Bell in California  194231        4
1  54315453  35.331677 -116.226027  Chris Bell in California  194231        3
2  54315455  35.359924 -116.178285  Chris Bell in California  194231        3
3  54315457  35.362457 -116.173913  Chris Bell in California  194231        3
```

```
4  54315458  35.364977 -116.169689  Chris Bell in California  194231        3

   changeset             timestamp        id    key  value type
0    5929701  2010-10-01T17:14:39Z  54315452  power  tower  tag
1    5929701  2010-10-01T17:14:53Z  54315453  power  tower  tag
2    5929701  2010-10-01T17:14:35Z  54315455  power  tower  tag
3    5929701  2010-10-01T17:13:35Z  54315457  power  tower  tag
4    5929701  2010-10-01T17:14:52Z  54315458  power  tower  tag
```

Now count the number of power towers in Nodes dataset

[10]:
```
query = '''select count(*) as 'Number of Power Towers' from nodes n
inner join nodes_tags nt on nt.id = n.id
where nt.key = 'power'
limit 10000;'''
return_sql_query_as_dataframe(query).head()
```

[10]:
```
   Number of Power Towers
0                    3675
```

Improvements/Suggestions for Dataset

Problem: Lack of a large amount of useful information for Henderson citizens Henderson is home to several restaurants and fun places to eat. However, this dataset does not contain a tremendous amount of data for restauraunts. As mentioned in the challenges faced with the dataset, the types of users entering data into OSM are likely doing so for a purpose (professional/projects/etc.).

[19]:
```
query = '''SELECT value as 'Type of Food', count(*) as 'Number in Dataset'
FROM nodes_tags WHERE key='cuisine' group by value order by count(*) desc;'''
return_sql_query_as_dataframe(query)
```

[19]:
```
                                  Type of Food  Number in Dataset
0                                       burger                  5
1                                     sandwich                  4
2                                     american                  2
3                                   coffee_shop                 2
4                                         juice                 2
5                                         pizza                 2
6                                      Hawaiian                 1
7                                       chinese                 1
8                                     ice_cream                 1
9   international;donut;breakfast;tea;coffee_shop               1
10                                 italian;pizza                1
11                                      mexican                 1
12                                         thai                 1
13                                        wings                 1
```

Potential Solution: Leverage Yelp API The Yelp API already exists and has far more data pertaining to Henderson restaurants than what is available in OSM. A series of scripts could be incorporated to search within a target region against the Yelp API. Once the addresses are returning (assuming the Yelp API does not provide Lat/Long data), the Google Geocoding Service API could be implemented to obtain Lat/Long/other Geodata.

Problem: Lack of variety in users editing the dataset There is not a lot of variety in the dataset. This is because there is not a lot of variety in the users. To demonstrate this, all users and their contributions to the dataset as a percentage of the total dataset are shown below.

[34]:
```
query = '''SELECT user, count(user), ROUND(CAST(count(user) as float)/
 ↪CAST(353779 as float)*100,2) as "Percent of Data Created"
from nodes group by user having ROUND(CAST(count(user) as float)/CAST(353779 as␣
 ↪float)*100,2) > 1 order by count(user) desc;'''
user_df = return_sql_query_as_dataframe(query)
user_df
```

[34]:

	user	count(user)	Percent of Data Created
0	alimamo	134304	37.96
1	Stephen Show	102666	29.02
2	Jwheels9876	16477	4.66
3	Tom_Holland	7493	2.12
4	CueVii	7458	2.11
5	woodpeck_fixbot	6911	1.95
6	TheDutchMan13	5775	1.63
7	knutella	5010	1.42
8	Victoria1901	5005	1.41
9	Seandebasti	4493	1.27
10	ukrainian_falcon	4121	1.16

[35]:
```
user_df['Percent of Data Created'].sum()
```

[35]:
```
84.71
```

As seen above, 11 users are responsible for over 84% of the data in Henderson. There are over 350,000 records in the dataset. This is an astounding rate. One user (alimamo) has over 38% of the data alone.

Potential Solution: User/Citizen Education Because the vast majority of Henderson citizens are not aware of OSM, the pool of citizens that have the knowledge and ability to update/enter data is very small. Perhaps as a community service a monthly class could be held at local libraries to teach citizens how to correctly enter data and what kind of data would be useful/interesting to users of OSM. This could also create interest in the GIS field for students/citizens entering the career field!