

DATA WRANGLING USING SQL PROJECT

1. INTRODUCTION AND MOTIVATION:

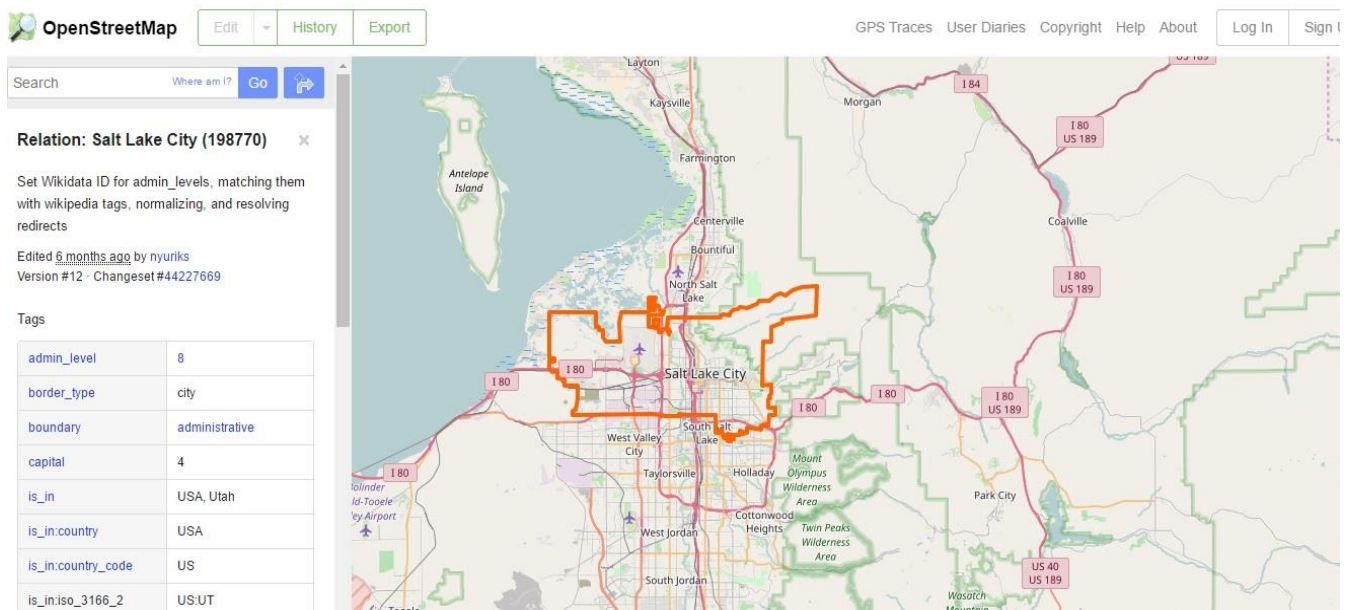


Figure 1: Salt Lake City Boundary on Map

For this project, Salt Lake City is selected for analysis. The reason behind selection for this particular city is that I am currently living in this beautiful city. Being fairly new to the city, I was curious to explore more of this region and gather some observations. Following are key points of this project:

- Data auditing, cleaning and parsing using python 2.7 (Spyder)
- Data analysis using SQL
- Program for analysis done in IPython (Jupyter)
- Creating appropriate csv files and querying them
- Five different observations made from the data using SQL
- Two Problems present in the dataset explored
- Improvements and general comments mentioned

2. DATA PREPARATION PROCESS:

In order to explore the data, converting it into appropriate format is necessary. The data for this region is downloaded from open street map website, which is an '.osm' file. Following files are created in python which prepares the data for further analysis:

- Audit.py:

The names of street are not in uniform format. On some occasions, short form is used which might create problems during investigation. In order to clean this type of data, a list of properly formatted words called 'expected' is created. Whenever the address tag words are not in this format, they will be substituted by words defined in 'mapping' list accordingly. (Only the last words of the tags are replaced) Functions for these operations are defined in this file.



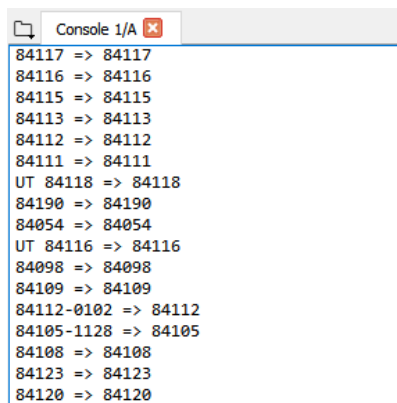
```
Console 1/A
W 400 S => W 400 South
Parleys Way #240 => Parleys Way #240
2700 W => 2700 West
300 W => 300 West
700 W => 700 West
1955 W => 1955 West
S 3200 W => S 3200 West
West Parkway Boulevard Suite 100 => West Parkway Boulevard Suite 100
239 S Main St => 239 S Main Street
Orange St => Orange Street
S State St => S Streetate Street
Exchange Place, Suite 60 => Exchange Place, Suite 60
S. 400 W. => S. 400 West
Komas => Komas
University Village => University Village
West Legacy Crossing Blvd => West Legacy Crossing Boulevard
Hope Ave => Hope Avenue
History log  IPython console  Python console
```

Figure 2: Address Formatting

Another data type which needs to be cleaned is the postal codes. Postal codes indicate the area code of the region in the city. Most of the codes are present in the form of 5 digits. However, some of the entries do not follow this format. Some of them are:

1. UT XXXXX
2. XXXXX-XXXX

Such entries are formatted to only 5 digits by placing condition for cleaning on each pattern.



```
Console 1/A
84117 => 84117
84116 => 84116
84115 => 84115
84113 => 84113
84112 => 84112
84111 => 84111
UT 84118 => 84118
84190 => 84190
84054 => 84054
UT 84116 => 84116
84098 => 84098
84109 => 84109
84112-0102 => 84112
84105-1128 => 84105
84108 => 84108
84123 => 84123
84120 => 84120
```

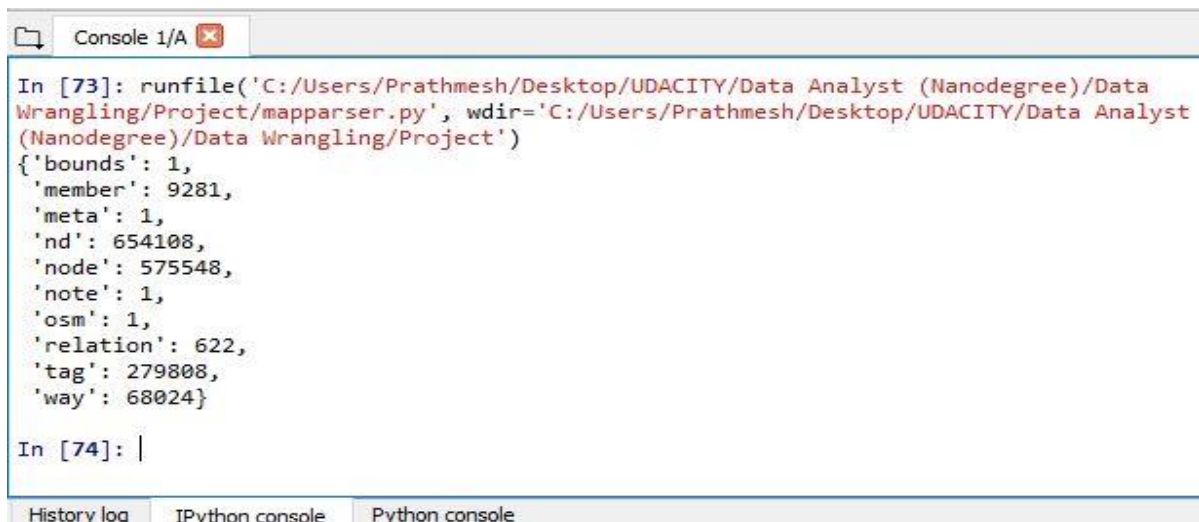
Figure 3: Postal Codes Formatting

- Data.py:

This file cleans all the formatting problems discussed in the previous section, explores (parses) the '.osm' file and creates appropriate csv files for the same. In the first part of the file, all the data auditing functions are included. Name of the csv files as well as their fields are also defined here. Another function 'shape_element' is defined which helps us in parsing the data. This program firstly checks whether the tag is a 'node' or 'way'. Then, depending upon format of the tag, (PROBLEMCHARS: problematic characters or LOWER_COLON: colon present in the names) these are saved separately. Finally, these are written in the earlier defined csv files. The files for analysis are ready now.

- Mapperparser.py:

This file is used for determining the number of tags present in the data. It contains a simple for loop which parses the '.osm' file and returns number of tags grouped. Running this file gives us this data:

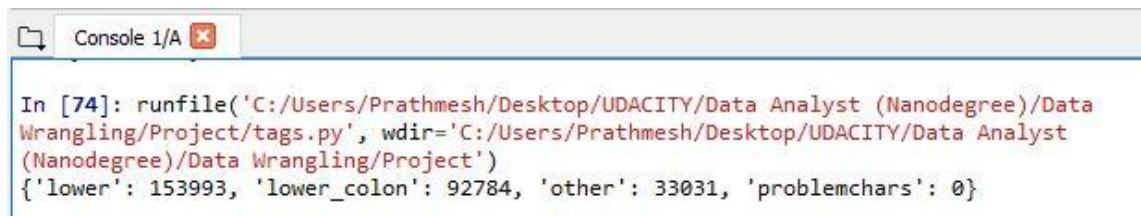


```
Console 1/A [X]
In [73]: runfile('C:/Users/Prathmesh/Desktop/UDACITY/Data Analyst (Nanodegree)/Data Wrangling/Project/mapparser.py', wdir='C:/Users/Prathmesh/Desktop/UDACITY/Data Analyst (Nanodegree)/Data Wrangling/Project')
{'bounds': 1,
 'member': 9281,
 'meta': 1,
 'nd': 654108,
 'node': 575548,
 'note': 1,
 'osm': 1,
 'relation': 622,
 'tag': 279808,
 'way': 68024}
In [74]: |
```

Figure 4: Map parser file results

- Tags.py:

This file groups together the tags in the dataset by their format. No problematic character tags were found out. The result returned by this program is:



```
Console 1/A [X]
In [74]: runfile('C:/Users/Prathmesh/Desktop/UDACITY/Data Analyst (Nanodegree)/Data Wrangling/Project/tags.py', wdir='C:/Users/Prathmesh/Desktop/UDACITY/Data Analyst (Nanodegree)/Data Wrangling/Project')
{'lower': 153993, 'lower_colon': 92784, 'other': 33031, 'problemchars': 0}
```

Figure 5: Tags file results

- Users.py:

This file is used to determine the number of different contributors for this area. When it is executed against our area of interest, it gives **698** number of contributors. This number is high, which indicates that Salt Lake City is fairly large and metro city.

- Schema.py:

This file contains all the information for the csv files which are needed to be created for further analysis. These 5 different files are named as:

1. Node.csv
2. Node_Tags.csv
3. Way.csv
4. Way_Nodes.csv
5. Way_Tags.csv

3. DATA ANALYSIS & QUERYING:

- File Sizes:

Once all the csv files are created, these are inserted in to a database file using SQL. This block of code provides all file sizes once they are created.

```
# Displaying sizes of all project files:

import os

print "OSM File Size is:", os.path.getsize('SLC.osm')*1e-6, "MB"
print "project.db file is:", os.path.getsize('project.db')*1e-6, "MB"
print "nodes.csv file is:", os.path.getsize('nodes.csv')*1e-6, "MB"
print "nodes_tags.csv file is:", os.path.getsize('nodes_tags.csv')*1e-6, "MB"
print "ways.csv file is:", os.path.getsize('ways.csv')*1e-6, "MB"
print "ways_nodes.csv file is:", os.path.getsize('ways_nodes.csv')*1e-6, "MB"
print "ways_tags.csv file is:", os.path.getsize('ways_tags.csv')*1e-6, "MB"

OSM File Size is: 127.842843 MB
project.db file is: 72.327168 MB
nodes.csv file is: 48.703848 MB
nodes_tags.csv file is: 1.998708 MB
ways.csv file is: 4.128661 MB
ways_nodes.csv file is: 16.130337 MB
ways_tags.csv file is: 8.107374 MB
```

Figure 6: File size code

As you see that the OSM file size is well above 50 MB. (which was the requirement) The other csv files are considerably smaller in size.

- Number of nodes and ways present:

Firstly, all the tables are created in the 'project.db' file according to the details mentioned in the schema file. These two blocks of code provide us the number of nodes and ways in the dataset. We can see that number of 'nodes' are significantly higher than 'way'.

```
: # 2. Getting number of nodes in the table:

cur.execute("SELECT COUNT(*) FROM nodes;")
print(cur.fetchall())

[(575548,)]
```

Figure 7: Node Number Query

```
: # 4. Getting number of ways in the table:

cur.execute("SELECT COUNT(*) FROM ways;")
print(cur.fetchall())

[(68024,)]
```

Figure 8: Way Number Query

- Analysis 1 - Number of unique users and most common ones:

This block of code returns number of different users and also most common contributors to this map. From the results, we can see almost same numbers which we acquired from 'users.py' file. There are 692 unique contributors with 'chadbunn' being most common one by a huge margin.

```
# Analysis 1: Total Number of Unique Users and 10 Most Common Ones:

cur.execute("SELECT COUNT(DISTINCT(e.uid)) \
            FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;")

print(cur.fetchall())

cur.execute("SELECT e.user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e \
            GROUP BY e.user \
            ORDER BY num DESC LIMIT 10;")

print(cur.fetchall())

[(692,)]
[('chadbunn', 244599), ('butlerrn', 69490), ('mvexel', 50240), ('wrk3', 37352), ('mash84121', 31834), ('woodpeck_fixbot', 2295), ('nemmer', 16151), ('balcoath', 13526), ('TheDutchMan13', 11184), ('Val', 9913)]
```

Figure 9: Analysis 1 Results

- Analysis 2 – Distribution of religion in the city:

Christianity is the most followed religion in this country. This may vary state to state, but before looking at the result, it is expected that this trend may follow in this region. All querying the dataset, expected result was given. Christian religion is followed by majority of people.

```
# Analysis 2: Distribution of Religion in the City:

import pprint
cur.execute ("SELECT tags.value, COUNT(*) as num FROM (SELECT * FROM nodes_tags UNION ALL \
            SELECT * FROM ways_tags) tags \
            WHERE tags.key LIKE '%religion'\
            GROUP BY tags.value \
            ORDER BY num DESC;")

pprint.pprint(cur.fetchall())

[('christian', 532),
 ('muslim', 3),
 ('buddhist', 2),
 ('jewish', 1),
 ('scientologist', 1),
 ('unitarian_universalist', 1)]
```

Figure 10: Analysis 2 Results

- Analysis 3 – Drinking Bars in the City:

General opinion about this city states that drinking rules are pretty strict and availability of liquor is very limited. Therefore, I wanted to explore further if that is the reality. For this purpose, a query is constructed which returns total number of bars present in the city. From this query, we know that total **45** drinking bars are there. Also, these places can also be named differently. (e.g. pub) So, the liquor situation is not as bad as believed!

```
# Analysis 3: Number of drinking bars in the City:

cur.execute("SELECT COUNT(*) FROM nodes_tags WHERE value LIKE '%bar';")
print(cur.fetchall())

[(45,)]
```

Figure 11: Analysis 3 Results

- Analysis 4 – Amenities available in the City:

Significant amount of information can be extracted by looking at amenities present in the city. Since, these amenities are made available to public, different number of these can be helpful in constructing some observations about the city. For this analysis, all the amenities present in the city are extracted and presented in descending order.

```
# Analysis 4: Overview of Amenities Available in the City:

import pprint
cur.execute ("SELECT tags.value, COUNT(*) as num FROM (SELECT * FROM nodes_tags UNION ALL \
SELECT * FROM ways_tags) tags \
WHERE tags.key LIKE '%amenity'\
GROUP BY tags.value \
ORDER BY num DESC;")

pprint.pprint(cur.fetchall())

[('parking', 1020),
('restaurant', 709),
('place_of_worship', 540),
('fast_food', 294),
('school', 283),
('fuel', 141),
('bank', 124),
('cafe', 102),
('bench', 99),
('toilets', 68),
('drinking_water', 50),
('bar', 37),
('shelter', 36),
('bicycle_parking', 34),
('bicycle_rental', 33),
('parking_entrance', 28),
('library', 27),
('post_office', 27),
('fire_station', 26),
('pharmacy', 24),
('theatre', 22),
('atm', 20),
('car_wash', 20),
('fountain', 19),
('hospital', 18),
('cinema', 16),
('pub', 16),
('post_box', 15),
('vending_machine', 13),
('clinic', 11),
('doctors', 10),
('grave_yard', 10),
('public_building', 10),
('university', 10),
('dentist', 9),
```

Figure 12: Analysis 4 Results:

From this result, many different statements can be made. High number of restaurants implies the people are very outgoing and enjoy hotel food. Majority of people in this city are religious in nature (Christianity) as evident from large number of places of worship. Bicycles are also frequently used by people. The planning of the city also seems to be good since large areas for parking are provided in the city.

- Analysis 5 – Most common shops in the City:

In order to further explore the city, we take a look at the shops present in the city. As this list is large in number, we limit this result to first 25 different types of shops. After querying the data, we get expected results with clothes shops being most in number. Other shops like supermarket, shoes, beauty, etc are also present pretty high in the list. The second most common shop 'convenience' indicates that people like to buy multiple daily items at one store.

```
# Analysis 5: 25 Most Common Shops in the City:

import pprint
cur.execute ("SELECT value, COUNT(*) as num FROM nodes_tags \
            WHERE key = 'shop'\
            GROUP BY value \
            ORDER BY num DESC LIMIT 25;")

pprint.pprint(cur.fetchall())

[('clothes', 118),
 ('convenience', 66),
 ('yes', 48),
 ('supermarket', 46),
 ('hairstylist', 44),
 ('car_repair', 39),
 ('department_store', 36),
 ('beauty', 35),
 ('mobile_phone', 31),
 ('car', 27),
 ('jewelry', 24),
 ('shoes', 23),
 ('bicycle', 20),
 ('books', 20),
 ('bakery', 16),
 ('furniture', 15),
 ('deli', 13),
 ('dry_cleaning', 12),
 ('gift', 12),
 ('electronics', 11),
 ('copyshop', 10),
 ('variety_store', 10),
 ('sports', 9),
 ('car_parts', 8),
 ('tyres', 8)]
```

Figure 13: Analysis 5 Results:

- Problem in the dataset – Phone Numbers:

Phone numbers are usually 10 digits in length. Upon retrieving the list of numbers available, we see mixed formats present. Some of them are:

1. xxx-xxx-xxxx
2. (xxx) xxx-xxxx
3. xxx.xxx.xxxx
4. +1 xxx xxx xxxx
5. +1-xxx-xxx-xxxx
6. xxxxxxxxxx

In order to clean this dataset further, these numbers must be converted into single format. This can be achieved by following similar procedure to one used in postal code cleaning. All the digits can be extracted, leaving additional characters like '-', '+1' and then substituting with only these 10 digits. This block of code generates the phone numbers available in the data:

```
# Dataset Problem 1: Phone Numbers

import pprint
cur.execute("SELECT value FROM ways_tags \
            WHERE key = 'phone';")

pprint.pprint(cur.fetchall())

[('801-396-9625',),
 ('435-527-5585',),
 ('(801) 447-6860',),
 ('801.575.2345',),
 ('(801) 262-4653',),
 ('801-466-8751',),
 ('801-483-5420',),
 ('801-250-6396',),
 ('(801) 238-7300',),
 ('801.533.4527',),
 ('801-535-6110',),
 ('801-539-0852',),
 ('801-524-8200',),
 ('+1 801 524 8100',),
 ('+1-801-532-5501',),
 ('(801) 973-6271',),
 ('385-468-1500',),
 ('801.583.9513',),
 ('801-581-6326',),
 ('+1 801 662 1000',),
 ('+1 801 587 7000',),
 ('+1 801 536 3500',),
 ('801-298-6040',),
 ('(801) 966-4653',),
 ('801) 266-8621',),
 ('385-468-1400',),
 ('801-414-4103',),
 ('+1 801 964 3100',),
 ('+1 801 328 3288',),
 ('+1-801-487-7736',),
 ('+1-385-468-1305',),
 ('1-801-539-8888',),
 ('(801) 484-7651',),
 ('385-468-1440',),
```

Figure 14: Problem 1 in dataset

4. GENERAL COMMENTS (BENEFITS & ISSUES):

A. ISSUES:

1. An attempt to clean and analyse the dataset for Salt Lake City is made. After exploring some the relations from the data, it is evident that the data is cleaned enough for study at this level. However, in order to use this data to even greater extent, (like creating mobile apps using this data) more cleaning and formatting will be necessary.
2. Some of the future changes that can be done are related to phone numbers. We have replaced all the last words in the address tags to standard format and converted postal codes into 5 digits. This can be extended to whole address string. Further, as explored in the problem section, phone number needs to be cleaned to a particular pattern if it is included in the analysis.

B. BENEFITS:

1. By creating separate python files for cleaning and parsing, each step of the project can be viewed separately. SQL queries makes analysis using database tables much simpler to understand. Overall, this makes the project more accessible.
2. Great insights and observations about a city can be made using this project. Also, by replacing the data file for this region to another one, we can make similar analysis for that new city without making any major changes to the code.

5. REFERENCES:

1. www.openstreetmap.org
2. Udacity Data Wrangling Course (Case Study)