WRANGLING OpenStreetMaps DATA WITH SQL

Name: Jesus Fandino

MAP AREA: Tampa FL United States.

MapZen URL: https://mapzen.com/data/metro-extracts/metro/tampa_florida/

OSM XML 23 MB file, direct link:
https://s3.amazonaws.com/metro-extracts.mapzen.com/tampa_florida.osm.bz2

OpenStreetMaps: http://www.openstreetmap.org/search?query=Tampa%2C%20FL#map=10/27.8348/-82.2931

I selected this extract because it contains the city where I live.

Problems encountered in the .osm file:

I. Inconsistencies in zip codes:

- Zip codes with nine digits (i.e. 33613-4649) which are correct, but they are just a few in comparison to the standard five-digit codes that will result in a different outcome when querying for zip codes.
- Zip codes with the standard format that do not belong to the area of interest. One found (35655).
- Two zip codes separated by a colon (33701:33704). This kind of format could be valid for ways (tiger:zip_left, tiger:zip_right) but not for nodes.
- Zip codes starting with state letter code (i.e. FL 34236)
- Missing values. One found (FL)

We wrote the function fix_zipcodes to handle all of those problems. However, we did not attempt to solve the inconsistencies in tiger:zip_left or tiger:zip_right tags.

```
def fix_zipcodes(zipcodes):
    '''Fix zip codes of the form 3####-#...#, 3####:#...# or Fl 3####, with the number
    following the 3 being just 3 or 4, according with the zip codes of Tampa area. The output
    is just a 5 digits string. If any other pattern is detected, the function returns FIXME'''

    fixed_zip = 'FIXME'
    pattern1 = re.compile(r'^3[3-4]\d\d\d')
    pattern2 = re.compile(r'^FL\s+3[3-4]\d\d\d$',  re.IGNORECASE)
    if re.search(pattern1, zipcodes):
        fixed_zip = zipcodes[:5]
    elif re.search(pattern2, zipcodes):
        fixed_zip = zipcodes[zipcodes.find('3'):]
    elif zipcodes == '35655': # A typo found for a zip code related to Trinity, FL.
        fixed_zip = '34655'
    return fixed_zip
```

- By checking out the integrity of the postal_code key with the aid of the following query, we found a misplaced value:

```
db = sqlite3.connect(database)
query = """SELECT value as zip_code, COUNT(*) as occurrences FROM ways_tags
        WHERE key = 'postal_code' GROUP BY value;"""
zipcodes_df = pd.read_sql_query(query, db)
db.close()
zipcodes_df
```

	zip_code	occurrences
0	(813) 643-1700	1
1	33565	1
2	34219	21
3	34222	3

It seems like a phone number was introduced by mistake in the field belonging to zip codes.
We corrected this issue by moving this value to the corresponding 'phone' key in the dictionary.

II. Inconsistencies found in street names:

- Simplified street names (i.e. 51st St W)
- Different names for the same way in the State Road System (FL 54, SR 54, State Road 54, West Brandon Blvd (S.R. 60))
- Non-unique name pattern for ways in the US Highway System (US Hwy 19, US 19, U.S. 19, US Highway 19, US-19)
- Suite numbers as part of the street names (Ulmerton Rd, Suite 107)
- Home numbers included in the street names (12000 US Highway 92)
- Street names containing foreign language (2300 Bee Ridge Rd, Sarasota, FL 34239, Vereinigte Staaten)
- Street names containing portions of the mailing address (1400 1st Ave W, Bradenton 34205)

That represents a broad spectrum of issues and demands a lot of work. We wrote some functions in an attempt to address those problems. However, it was not possible to solve all of them in the timeframe of this project. In spite of this, and after applying all the implemented cleaning procedures for street names, just a small fraction of them remained without a cleaning solution.

Also, we believe that it could be useful to have the street names without Cardinals at the beginning or the end. That allows us to find a given street (East Kennedy Boulevard, West Kennedy Boulevard or Kennedy Boulevard) without ambiguity. We did not strip Cardinals from the names in addr:street because that is a valuable information; we created a new key (u_street) in the dictionary and incorporated those entries into the database instead. That proved to be useful under certain kind of queries.

III. City names

- Non-unique city names: (Saint Petersburg, St. Petersburg)

IV. County names

- Non-unique county names (i.e., Pinellas and Pinellas, FL)

- More than one item in the place for County name (Manatee, FL; Polk, FL:Polk, FL)

V. Secondary tag attributes 'k' that belong to the PROBLEMCHARS group.

We were instructed to discard the data that belongs to this category. However, we found out that there were just three entries in our dataset that matched the PROBLEMCHARS pattern, all of them sharing issues of a particular kind (space characters within the name) and we decided to make a fix for those names allowing the inclusion of the data related into our database.

VI. Secondary tags containing 'census:population' and 'population' displayed some duplicated information.

We separated the year of the census from the population data in census:population and created a new key in the dictionary (census) to avoid the overwriting of the key 'population' under the rules applied for secondary tags containing ':' chars.

Overview of the data

We used SQL queries with the aid of the sqlite3 and pandas modules in Python to gather information about our database.

- File sizes:

Using python hurry.filesize:

```
import sqlite3
import pandas as pd
database = "TampaFlorida.db"
from pprint import pprint
import os
from hurry.filesize import size
dirpath = '/Users/jedfarm/Documents/UDACITY/P3'
files_list = []
for path, dirs, files in os.walk(dirpath):
    files_list.extend([(filename, size(os.path.getsize(os.path.join(path, filename))))
                       for filename in files])
for filename, size in files_list:
    if not filename.startswith('.'):
        print '{:.<40s}: {:5s}'.format(filename,size)
```

```
nodes.csv................................: 120M
nodes_tags.csv...........................: 6M
tampa_florida.osm........................: 330M
TampaFlorida.db..........................: 184M
ways.csv.................................: 9M
ways_nodes.csv...........................: 40M
ways_tags.csv............................: 30M
```

- Number of nodes and ways:

```
db = sqlite3.connect(database)
query = """SELECT COUNT(*) as nodes FROM nodes;"""
nodes_df = pd.read_sql_query(query, db)
db.close()
nodes_df
```

	nodes
0	1540285

```
db = sqlite3.connect(database)
query = """SELECT COUNT(*) as ways FROM ways;"""
ways_df = pd.read_sql_query(query, db)
db.close()
ways_df
```

	ways
0	170011

- Number of unique users

```
db = sqlite3.connect(database)
query = """SELECT COUNT(DISTINCT(e.uid)) as users
FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;"""
users_df = pd.read_sql_query(query, db)
db.close()
users_df
```

	users
0	1183

- Top ten contributors

```
db = sqlite3.connect(database)
query = """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;"""
bigusers_df = pd.read_sql_query(query, db)
db.close()
bigusers_df
```

	user	num
0	coleman	257212
1	woodpeck_fixbot	238662
2	grouper	185860
3	EdHillsman	109107
4	NE2	74517
5	David Hey	61182
6	LnxNoob	60261
7	westampa	42471
8	bot-mode	39428
9	Chris Lawrence	28280

- Amenities by number, top 10.

```
db = sqlite3.connect(database)
query = """SELECT value, COUNT(*) as num FROM nodes_tags WHERE key='amenity' GROUP BY value
        ORDER BY num DESC LIMIT 10;"""
amenities_df = pd.read_sql_query(query, db)
db.close()
amenities_df
```

	value	num
0	restaurant	810
1	place_of_worship	773
2	school	566
3	fast_food	383
4	bicycle_parking	354
5	bench	238
6	fuel	236
7	fountain	165
8	bank	163
9	fire_station	149

Other ideas and future work

- Streets with more restaurants (top 10):

When we made a query just asking for 'street', this is the result we obtained:

```
db = sqlite3.connect(database)
query = """SELECT nodes_tags.value as street, COUNT(*) as restaurants FROM nodes_tags JOIN
(SELECT DISTINCT(id), value FROM nodes_tags WHERE key='amenity' and value = 'restaurant') i
ON nodes_tags.id = i.id
WHERE nodes_tags.key = 'street' GROUP BY nodes_tags.value ORDER BY restaurants DESC LIMIT 10;"""
df = pd.read_sql_query(query, db)
db.close()
df
```

	street	restaurants
0	Central Avenue	22
1	4th Street North	18
2	East Fletcher Avenue	16
3	Ulmerton Road	11
4	East Fowler Avenue	10
5	Gulf to Bay Boulevard	9
6	West Hillsborough Avenue	9
7	McMullen Booth Road	8
8	East 7th Avenue	7
9	East Bay Drive	7

Apparently, there is nothing wrong with this table, but we suspected that leading and trailing Cardinals in street names might have been overshadowing the actual results. That propelled us to create the u_street key in the dictionary as mentioned above. Then we made a query similar to the last one but in this case using the u_street key:

```
db = sqlite3.connect(database)
query = """SELECT nodes_tags.value as street, COUNT(*) as restaurants FROM nodes_tags JOIN
(SELECT DISTINCT(id), value FROM nodes_tags WHERE key='amenity' and value = 'restaurant') i
ON nodes_tags.id = i.id WHERE nodes_tags.key = 'u_street' GROUP BY nodes_tags.value
ORDER BY restaurants DESC LIMIT 10;"""
df = pd.read_sql_query(query, db)
db.close()
df
```

	street	restaurants
0	Central Avenue	22
1	4th Street	20
2	Fletcher Avenue	17
3	Hillsborough Avenue	11
4	McMullen Booth Road	11
5	Ulmerton Road	11
6	Fowler Avenue	10
7	Kennedy Boulevard	10
8	Gulf to Bay Boulevard	9
9	1st Avenue	8

As a result, all the streets except for the first one (Central Avenue) experienced a change in the number of restaurants or their rank or both.

- By watching the area selected under the name of tampa_florida from MapZen, it was evident that it covers much more than just the city of Tampa. We were curious about which counties were included in this metro-extract.

```
db = sqlite3.connect(database)
query = """SELECT tags.value as county, COUNT(*) as count FROM (SELECT * FROM nodes_tags
        UNION ALL SELECT * FROM ways_tags) tags WHERE tags.key='county' or tags.key ='county1'
        GROUP BY tags.value ORDER BY count DESC;"""
counties_df = pd.read_sql_query(query, db)
db.close()
counties_df
```

	county	count
0	Hillsborough	23879
1	Pinellas	20808
2	Manatee	9823
3	Pasco	9072
4	Sarasota	4739
5	Polk	982
6	Hardee	41
7	Sumter	4
8	DeSoto	2

Our extract contains data from nine different counties, lead by Hillsborough and Pinellas. Also, the overall contribution from Hardee, Sumter and DeSoto is minuscule.

- As I moved to the Tampa Bay area just recently, I was curious about the demographics:

```
db = sqlite3.connect(database)
query = """SELECT nodes_tags.value as place, i.value as population FROM nodes_tags JOIN
(SELECT DISTINCT(id), value FROM nodes_tags WHERE key='population') i ON nodes_tags.id = i.id
WHERE nodes_tags.key = 'name'ORDER BY i.value *1 DESC LIMIT 10"""
pop_df = pd.read_sql_query(query, db)
db.close()
pop_df
```

	place	population
0	Tampa	332888
1	St. Petersburg	244324
2	Clearwater	107742
3	Brandon	103483
4	Largo	73796
5	Riverview	71050
6	Palm Harbor	57439
7	Bradenton	53662
8	Sarasota	52942
9	Pinellas Park	47354

The fact that the first seven places in this table belong to Hillsborough or Pinellas County explains the trend in counts in the preceding table.

- How active has been the OpenStreetMaps community in the area?

```
db = sqlite3.connect(database)
query = """SELECT strftime('%Y', t.timestamp) as Year, COUNT(*) as changes
        FROM (SELECT nodes.timestamp FROM nodes UNION ALL SELECT ways.timestamp FROM ways) t
        GROUP BY Year ORDER BY Year DESC;"""
df = pd.read_sql_query(query, db)
db.close()
df
```

	Year	changes
0	2016	139706
1	2015	166440
2	2014	137167
3	2013	313122
4	2012	276216
5	2011	167446
6	2010	131410
7	2009	366881
8	2008	8936
9	2007	2972

To make a little more complex queries, we created a VIEW:

```
db = sqlite3.connect(database)
mydb = db.cursor()
mydb.execute("""CREATE VIEW myview AS SELECT e.user, strftime('%Y', e.timestamp) as Year,
COUNT(*) as num
FROM (SELECT user, timestamp FROM nodes UNION ALL SELECT user, timestamp FROM ways ) e
GROUP BY e.user, strftime('%Y', e.timestamp) ORDER BY num DESC,
strftime('%Y', e.timestamp) DESC""")
mydb.close()
```

Now, myview allows us to write more readable queries for a number of questions, for example:

- How many users contributed each year?
- Is there a contributor (or more) that has been active every year since 2007?
- If the previous question has a none as an answer, then who is the most veteran among all active contributors?
- Who are the top contributors on each of the past ten years?

To illustrate this, we choose to answer the last of the above questions:

```
db = sqlite3.connect(database)
query = """SELECT user, myview.Year, num as changes FROM myview
           INNER JOIN (SELECT Year, MAX(num) AS maxnum FROM myview GROUP BY Year) q
           ON myview.Year = q.Year AND myview.num = q.maxnum ORDER BY myview.Year DESC;"""
top_df = pd.read_sql_query(query, db)
db.close()
top_df
```

	user	Year	changes
0	miluethi	2016	26678
1	coleman	2015	22392
2	coleman	2014	41011
3	coleman	2013	98511
4	coleman	2012	69674
5	EdHillsman	2011	35853
6	EdHillsman	2010	30021
7	woodpeck_fixbot	2009	224706
8	David Hey	2008	6433
9	DaveHansenTiger	2007	2939

Not many surprises here, almost all of the names in this table belong to the top ten contributors found previously. However, maybe we just have found an explanation for the sharp surge in changes observed in 2009 (see previous table): woodpeck_fixbot.

Future work

- To obtain more accurate results performing queries that involve zip codes in the future, it is necessary to clean up the TIGER data related to right and left zip codes. Very often those secondary tags contain explicit (;) or implicit (:) lists of zip codes. I believe one of the best choices to address this condition is to split up the list in as many new keys in the dictionary as necessary. That is not difficult to implement, but a decision has to be made in relation to adopt a name pattern for the keys.

- Further cleaning of the street names should include single case solutions, search for some characters (such us: '(', ')', '.', '-') within the street name and provide a cleaning solution for County Roads. Also, unique solutions have to be performed for cases of multiple-name streets. This is a far from a simple task, due to the fact that we will face the trade-off of given a street the most general name available, taking the risk of that name being meaningless for many potential users or vice-versa, giving that street the most common name, contributing to the current ambiguity. In any case, this is a decision that will take some research.

- Also, some basic analysis is recommended for fields such us phone numbers and others that we left unexplored in this project.

Conclusions

We performed data wrangling on the tampa_florida extract from OpenStreetMaps that went far beyond from finding and fixing five issues, as requested. However, there is still certain amount of work to be done before this dataset is 100% clean. Nevertheless, the current state of this data allowed us to answer a number of interesting questions, and potentially some more we did not even think about here.

Files

- audit_city_names.py
- audit_county_names.py
- audit_county_tags.py
- audit_population_tags.py
- audit street.py
- audit_street_suite.py
- audit_streetnames_num.py
- audit_streets_state_roads.py
- audit_tags.py
- audit_tag_types.py
- audit_us_highway_names.py
- audit zipcodes.py
- create_db.py …… Creates a database from .csv files
- create_sample_osm.py
- file_sizes.py
- get_element.py
- clean_data.py …… Creates .csv files from a .osm file
- make_a_view.py
- query_db.py …… Executes queries to the database
- references.txt
- sample.osm
- P3.html
- P3.pdf