OpenStreetMap Data Wrangling with SQL

William Dew

Map Area: San Diego CA, USA

https://mapzen.com/data/metro-extracts/metro/san-diego_california/https://www.openstreetmap.org/export#map=10/32.8259/-117.1074

Problems with the Dataset¶

While working with the data about San Diego I came across three main problems that I had to code around to clean up the data. I will discuss my process for each problem in the following order.

1. Abbreviated street names
2. Street names in second level of address
3. So big that I couldn't put it all in memory when building a tree
4. Running time of the Program was very long

Abbreviated Street Names

Looking at the street names there were a lot of different abriviations like 'Blvd', 'Blvd.', or 'Bl'. To make the address database easier to search for names of streets I created a function that would change the name during the html to sql database translation. The following is my code used:

```
# Dictionary of all the different abriviations of street names
mapping = { "Av": "Avenue",
        "Ave": "Avenue",
        "Blvd": "Boulevard",
        "Bl": "Boulevard",
        "Ct": "Court",
        "Ctr": "Center",
        "Dr": "Drive",
        "S": "South",
        "N": "North",
        "E": "East",
        "Ln": "Lane",
        "Ml": "Mill",
        "Pk": "Park",
        "Pl": "Place",
        "Py": "Parkway",
        "Rd": "Road",
        "W": "West",
```

```
            "street": "Street",
            "St": "Street",
            "st": "Street",
            "Pkwy": "Parkway",
            "Wy": "Way",
            "Ste": "Suite"
            }

def replace(match):
    return mapping[match.group(0)]


def fix_address(string):
    # splits each word into a list.
    wordList = re.sub("[^\w]", " ",  string).split()
    fixed = []
    # For each word in wordList the matching value in the mapping dictionary is
    # is joined to a string then appended to a list.
    for i in wordList:
        p = re.sub('|'.join(r'\b%s\b' % re.escape(s) for s in mapping),
                replace, i)
        fixed.append(p)
    # fixed is turned into a string and returned
    fixed =(" ").join(fixed)
    return fixed
```

Street names in a second level of the k value in tags

Inside the tags the k value started with addr: followed by what the address was for example tag k="addr:city" v="Chicago". This was true for street, zipcode, and house number amoung others. I needed a way to extract the key type from the tag to make sense of the value of the tag. I completed this with the following code in the html to sql database translation.

```
if "addr:" in k:
    # if the second level tag "k" value starts with "addr:", it
    # should be added to a dictionary "address"
    # if the second level tag "k" value contains problematic characters,
    # it should be ignored
    if is_tag_viable(tag):
        fixed_v = fix_address(v)
        nk = k.split(":")
        address[nk[1]] = fixed_v
        my_dict["address"] = address
```

So big that I couldn't put the whole tree into memory

When running my code I found that as the larger my html file was the larger my tree would become and eventually I would get an error stating that there was no more memory to use. This caused a problem because the file for San Diego was over 2G of data. I had to find a way to read parts of the data into a tree and then delete it. I accomplished this with the following code.

```
def process_map(file_in, pretty = False):
    file_out = "{0}.json".format(file_in)
    with codecs.open(file_out, "w") as fo:
        for elem in get_element(file_in):
            el = shape_element(elem)
            if el:
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
                elem.clear() # clears the working element so tree is not built
                        # in memory and bogs down computer
```

Running Time was very long

The running time of this program was very long. It took San Diego html file a couple of days to convert to a SQL database. This is something I will have to look into to figure out how I can cut this down. Maybe running chunks of data from the html or looking at my algorithms.

Database Queries

Size of Files

1. san-diego_california.osm 292300 KB
2. san-diego_california.osm.json 292300 KB
3. san_deigo Data Base File 190566 KB

```
import sqlite3

def create_connection(db_file):
    #create a database connection to the SQLite database specified by db_file
    # retuns connection if db was made or None if db was not.
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except IOError as e:
        print(e)
```

```
    return None
```

Number of Unique Users

```
conn = create_connection("san_diego.db")

query = "select count(distinct(uid)) from node"
num_nodes = conn.execute(query).fetchall()
print(num_nodes)
conn.close()
```

```
[(994,)]
```

Number of Nodes

```
conn = create_connection("san_diego.db")

query = "select count(*) from node"
num_nodes = conn.execute(query).fetchall()
print(num_nodes)
conn.close()
```

```
[(1042218,)]
```

Number of Way

```
conn = create_connection("san_diego.db")

query = "select count(*) from way"
num_nodes = conn.execute(query).fetchall()
print(num_nodes)
conn.close()
```

```
[(94867,)]
```

Top Amenities

```
conn = create_connection("san_diego.db")

query = "select value, count(*) as num from node_tag where key = 'amenity' group by value order by num desc limit 10;"
num_nodes = conn.execute(query).fetchall()
print(num_nodes)
conn.close()
```

```
[(u'place_of_worship', 916), (u'fast_food', 542), (u'restaurant', 500), (u'school', 298), (u'bar', 275), (u'cafe', 179), (u'fuel', 111), (u'bank', 83), (u'bench', 76), (u'drinking_water', 73)]
```

Distinct Zipcodes

```
conn = create_connection("san_diego.db")

query = "select count(distinct(value)) from address where key = 'postcode';"
```

```
num_nodes = conn.execute(query).fetchall()
print(num_nodes)
conn.close()
[(84,)]
```

Number of Pizza Restraunts

```
conn = create_connection("san_diego.db")

query = """select node_tag.value, count(*) as num from node_tag
join (select distinct(id) from node_tag where value='restaurant') i
on node_tag.id=i.id
where key = 'cuisine' and value = 'pizza'
group by node_tag.value
order by num desc;"""
num_nodes = conn.execute(query).fetchall()
print(num_nodes)
conn.close()
[(u'pizza', 29)]
```

Address of Pizza Restraunts

```
conn = create_connection("san_diego.db")

query = """select value from address
inner join (select distinct(id) from node_tag where value='pizza') i
on address.node_id=i.id
where key = 'street';"""
query_2 = """select value from address
inner join (select distinct(id) from node_tag where value='pizza') i
on address.node_id=i.id
where key = 'housenumber';"""
street = conn.execute(query).fetchall()
housenumber = conn.execute(query_2).fetchall()
address = []
for i in street:
    n = i[0]
    address.append(n)
full_address = []
for i, q in enumerate(housenumber):
    n = str(int(float(q[0])))
    a = n + ' ' + address[i]
    full_address.append(a)
print(full_address)
conn.close()
[u'1290 University Avenue', u'10251 Mast Boulevard', u'6780 Miramar Road', u'
8049 Winter Gardens Boulevard', u'4545 La Jolla Village Drive']
```

Suggestions for Improving Data

Some amenities don't have addresses tied to them. When I tried to find the address of a vegan restaurant the query came up blank. It happened to a couple of other amenities too. It seems users were excited to put in restaurant names and types but not where to find them.

```
conn = create_connection("san_diego.db")

query = """select value from address
inner join (select distinct(id) from node_tag where value='vegan') i
on address.node_id=i.id
where key = 'street';"""
num_nodes = conn.execute(query).fetchall()
print(num_nodes)
conn.close()
[]
```

One way of fixing this would be to cross reference the name of the amenity with google api or another database. Another way to fix this would be to have the users put in their favorite restaurant and address or comic shop and address. This would engage the user with passion for their favorite thing and get the addresses up to date.

An issue that could arise though is some amenities wouldn't get any love and not have their addresses updated or checked. A church for example may not be someone's favorite spot so the churches' address would not be checked.

Conclusion

The Data from San Diego is incomplete. Some addresses are missing from amenities, zipcodes and postcodes reference the same value. If we are able to tap into the users passions of amenities we could fix some of these issues. It makes the user a lot more involved. There is a lot of different ways that the San Diego open street map data can be improved and cleaned up if simple unifying data tropes were used.