OpenStreetMap Data Wrangling Project Report

Chosen Map Area: San Jose, CA, USA

Why: I live in the area and wanted to explore it in detail

Data source: https://mapzen.com/data/metro-extracts/metro/san-jose_california/ (https://mapzen.com/data/metro-extracts/metro/san-jose_california/)

Data Sampling & Auditing

I imported the data and created a sample file of 10MB size to test and audit the data. The sample file code is exactly the same as provided in the Udacity lecture notes, with a k value = 28.

I then audited the file using the data_audit.py script. The issues I found were:

- Street types were inconsistently formatted
- Street names had all sorts of short-forms and abbreviations in them
- Many of the postal codes had formatting issues

I look at these issues in more detail in the next section, with some code snippets

Data Issues In Detail

1. Inconsistently formatted Street Types

Street types had issues like the type 'Avenue' being present in the format 'ave', 'Ave.', 'Av', 'av.' etc. I formatted these types such that, for example

- Minto Dr => Minto Drive
- Saratoga Ave => Saratoga Avenue

In [ ]:
```
# List of expected street types
expected = ["Avenue", "Boulevard", "Circle", "Commons", "Court", "Drive", "Expressway",
            "Highway", "Lane", "Loop", "Parkway", "Place", "Plaza", "Real", "Road",
            "Row", "Square", "Street", "Terrace", "Walk", "Way"]

# Mapping of non-expected: expected street types
street_mapping = { "Ave": "Avenue", "ave": "Avenue", "Blvd": "Boulevard",
            "Boulevard": "Boulevard", "Boulvevard": "Boulevard",
            "Cir": "Circle", "Ct": "Court", "court": "Court",
            "Dr": "Drive", "Hwy": "Highway", "Ln": "Lane",
            "Mt.": "Mount", "Mt": "Mount","Rd": "Road",  "Rd.": "Road",
            "Sq": "Square", "st": "Street", "street": "Street", "St": "Street",
            "ste": "Suite", "ste.": "Suite", "Ste.": "Suite", "Ste": "Suite",
            "square": "Square", "parkway": "Parkway"
          }
```

2. Abbreviated/Badly Named Streets

Street names had issues like directions being written inconsistently - sometime abbreviated, sometimes not. Also, some street names had the state name in them or city name. I formatted these types such that, for example

- Zanker Rd., San Jose, CA => Zanker Road
- S. Bascom => South Bascom

In [ ]:
```
# Mapping of non-expected: expected directions & terms in street names
direction_unusual_mapping = {"E": "East", "E.": "East", "N": "North",
"N.": "North",
            "S": "South", "S.": "South", "W": "West", "W.": "West",
            "Pkwy": "Parkway", "Ste": "Suite", "Ave": "Avenue", "Rd":"
Road",
            "Mt": "Mount", "Mt.":"Mount", "rio":"Rio", "robles": "Robl
es"
            }

# Mapping of entire street names that are to be replaced
name_mapping = {"Zanker Rd., San Jose, CA": "Zanker Road", "Zanker Roa
d, San Jose, CA": "Zanker Road",
    "Ala 680 PM 0.1": "Interstate-680", "Hwy 17 PM 7.1": "Highway 17"
}
```

3. Inconsistently formatted Zip Codes

I found that zip/post codes had issues like the some having the state code 'CA' preceding them or having incomplete (less than 4 digit) PO box numbers. I formatted these types such that, for example

- 95014-321 => 95014
- CA 95014 => 95014

In [ ]:
```
''' Check for post codes'''
post_code_re = re.compile(r'^\d{5}(?:[-\s]\d{4})?$')

''' Check for characters only (no numbers)'''
all_chars_re = re.compile(r'^[a-zA-Z]+$')

''' Check for term 'CA' in the post codes'''
ca_in_zipcode_re = re.compile(r'^CA(.*)')
```

Data Update

I used update functions in the data_update.py file to update the street type, street names and zip code issues discussed above.

In [ ]:
```
for street_type, ways in street_types.iteritems():
    for name in ways:
        better_name = update_street_tags(name)
        print name, "=>", better_name
```

Gaundabert Ln => Gaundabert Lane Branham Ln => Branham Lane Mt Hamilton Rd => Mount Hamilton Road Linwood Dr => Linwood Drive Zanker Rd., San Jose, CA => Zanker Road S. Bascom => South Bascom N 1st street => North 1st Street Foxworthy Ave => Foxworthy Avenue Perivale Ct => Perivale Court

In [ ]:
```
for zip in postcodes:
    new_zip_code = update_zip_code(zip)
    print zip, "=>", new_zip_code
```

CA 95113 => 95113 95014-246 => 95014 CUPERTINO => None 95014-2143;95014-2144 => 95014 951251 => 95125 94087 => 94087

Data Processing for XML -> CSV conversion

I then used the functions in the data_processing.py script to create CSV tables for each of the nodes, nodes_tags, ways, ways_tags and ways_nodes XML data.

In this python script, I updated the provided shape_element function to use my above-defined update functions for street types, street names and zip codes. The rest of the code is used as-is from the Udacity course code.

Code snippet for the node tag shape element is below:

In [ ]:
```
def shape_element(element, node_attr_fields=NODE_FIELDS, way_attr_fiel
ds=WAY_FIELDS,
                  problem_chars=PROBLEMCHARS, default_tag_type='regula
r'):
    """Clean and shape node or way XML element to Python dict"""

    node_attribs = {}
    way_attribs = {}
    way_nodes = []
    tags = []

    if element.tag == 'node':
```

```
        for node in NODE_FIELDS:
            node_attribs[node] = element.attrib[node]

        for child in element:
            tag = {}

            if PROBLEMCHARS.search(child.attrib['k']):
                continue

            elif LOWER_COLON.search(child.attrib['k']):
                if child.attrib['k'] == "addr:street":
                        tag['id'] = element.attrib['id']
                        tag['key'] = child.attrib['k'].split(':', 1)[1
]
                        tag['value'] = update_street_tags(child.attrib
['v'])
                        tag['type'] = child.attrib['k'].split(':', 1)[
0]

                if child.attrib['k'] == "addr:postcode":
                    tag['id'] = element.attrib['id']
                    tag['key'] = child.attrib['k'].split(':', 1)[1]
                    tag['value'] = update_zip_code(child.attrib['v'])
                    tag['type'] = child.attrib['k'].split(':', 1)[0]
                else:
                    tag['id'] = element.attrib['id']
                    tag['key'] = child.attrib['k'].split(':', 1)[1]
                    tag['value'] = child.attrib['v']
                    tag['type'] = child.attrib['k'].split(':', 1)[0]


            else:
                tag['id'] = element.attrib['id']
                tag['key'] = child.attrib['k']
                tag['value'] = child.attrib['v']
                tag['type'] = 'regular'

            if tag:
                tags.append(tag)
        return {'node': node_attribs, 'node_tags': tags}
```

SQL Data Import

I then used the code written in the sql_data_import.py script to import the CSV files into SQL tables to use for further analysis

Data Analysis

File sizes

OSM FILES san-jose_california.osm 284.4 MB sample.osm 10.3 MB

CSV FILES nodes.csv 107.2 MB nodes_tags.csv 2.6 MB ways.csv 10.1 MB ways_tags.csv 21.4 MB ways_nodes.csv 35.5 MB

Number of Nodes

In [ ]:
```
cur.execute("SELECT COUNT(*) FROM nodes;")
pprint.pprint(cur.fetchall())
```

1285764

Number of Ways

In [ ]:
```
cur.execute("SELECT COUNT(*) FROM ways;")
pprint.pprint(cur.fetchall())
```

171032

Number of Unique Users

In [ ]:
```
cur.execute("SELECT COUNT(DISTINCT(all_users.uid)) \
             FROM \
                (SELECT uid FROM nodes \
                 UNION ALL \
                 SELECT uid FROM Ways) \
             all_users;")
pprint.pprint(cur.fetchall())
```

1245

Number of Amenities

In [ ]:
```
cur.execute("SELECT count(value) \
             FROM nodes_tags \
             WHERE key = 'amenity'")
pprint.pprint(cur.fetchall())
```

3749

Types of Amenities

In [ ]:
```
cur.execute("SELECT value, count(*) \
             FROM nodes_tags \
             WHERE key = 'amenity' \
             GROUP BY 1 ORDER BY 2 DESC LIMIT 10;")
pprint.pprint(cur.fetchall())
```

Looks like restaurants, cafes amd fast food amenities make up the majority of the 'amenity' type

[(u'restaurant', 766), (u'fast_food', 369), (u'cafe', 224), (u'place_of_worship', 186), (u'bicycle_parking', 175), (u'bench', 174), (u'school', 151), (u'toilets', 148), (u'fuel', 123), (u'bank', 116)]

Top food-related amenities by type

In [ ]:
```
cur.execute("SELECT nt1.value,nt2.value, count(*) \
            FROM nodes_tags nt1 \
            JOIN \
                (SELECT distinct(id), value \
                FROM nodes_tags \
                WHERE value in ('restaurant', 'cafe', 'bbq', 'fast_foo
d') \
                GROUP BY 1,2) nt2 \
            ON nt1.id = nt2.id \
            WHERE key = 'name' \
            GROUP BY 1,2 ORDER BY 3 DESC LIMIT 10;")
pprint.pprint(cur.fetchall())
```

Top result is Starbucks which is of the type 'cafe'.

In [ ]:
```
[(u'Starbucks', u'cafe', 71),
 (u'Subway', u'fast_food', 32),
 (u'Taco Bell', u'fast_food', 11),
 (u"McDonald's", u'fast_food', 10),
 (u'Panda Express', u'fast_food', 9),
 (u'Burger King', u'fast_food', 8),
 (u'Pizza My Heart', u'restaurant', 8),
 (u"Togo's", u'fast_food', 8),
 (u'Jamba Juice', u'fast_food', 7),
 (u'KFC', u'fast_food', 7)]
```

Top Food-related Amenity by Location and Type

In [ ]:
```
cur.execute("SELECT nt1.value, nt2.value, nt3.value, count(*) \
            FROM nodes_tags nt1 \
            JOIN \
                (SELECT distinct(id), value \
                FROM nodes_tags \
                WHERE value in ('restaurant', 'cafe', 'bbq', 'fast_foo
d') \
                GROUP BY 1,2) nt2 \
            ON nt1.id = nt2.id \
            JOIN \
                (SELECT distinct(id), value \
                FROM nodes_tags \
                WHERE key = 'city' \
                GROUP BY 1,2) nt3 \
            ON nt1.id = nt3.id \
            WHERE key = 'name' \
            GROUP BY 1,2,3 ORDER BY 4 DESC LIMIT 20;")
pprint.pprint(cur.fetchall())
```

Starbucks is the top results with the most locations in San Jose, Sunnyvale and Santa Clara

In [ ]:
```
[(u'Starbucks', u'cafe', u'San Jose', 7),
 (u'Starbucks', u'cafe', u'Sunnyvale', 3),
 (u"Applebee's", u'restaurant', u'San Jose', 2),
 (u'KFC', u'fast_food', u'San Jose', 2),
 (u"Lee's Sandwiches", u'restaurant', u'San Jose', 2),
 (u'Starbucks', u'cafe', u'San Jos\xe9', 2),
 (u'Starbucks', u'cafe', u'Santa Clara', 2),
 (u'Subway', u'restaurant', u'San Jose', 2),
 (u'A&W', u'fast_food', u'San Jose', 1),
 (u'ABC Seafood Restaurant', u'restaurant', u'Milpitas', 1),
 (u'Almaden Sushi', u'restaurant', u'San Jose', 1),
 (u"Amici's East Coast Pizzeria", u'restaurant', u'Cupertino', 1),
 (u'Aqui', u'restaurant', u'San Jose', 1),
 (u'Arka', u'restaurant', u'Sunnyvale', 1),
 (u'Asia Village', u'restaurant', u'Sunnyvale', 1),
 (u'B2 Coffee', u'cafe', u'San Jose', 1),
 (u'Baja Fresh', u'fast_food', u'San Jos\xe9', 1),
 (u'Bambu', u'restaurant', u'Sunnyvale', 1),
 (u'Banana Leaf', u'restaurant', u'Milpitas', 1),
 (u'Barn Thai', u'restaurant', u'Sunnyvale', 1)]
```

Top Cuisines

In [ ]:
```
cur.execute("SELECT nt1.value, COUNT(*) \
            FROM nodes_tags nt1 \
            JOIN \
                (SELECT distinct(id) as id \
                FROM nodes_tags \
                WHERE value = 'restaurant') nt2 \
            ON nt1.id = nt2.id \
            WHERE key = 'cuisine' \
            GROUP BY 1 ORDER By 2 DESC LIMIT 10")
pprint.pprint(cur.fetchall())
```

Top 3 cuisines in San Jose are Chinese, Vietnamese and Mexican

In [ ]:
```
[(u'chinese', 61),
 (u'vietnamese', 59),
 (u'mexican', 51),
 (u'pizza', 49),
 (u'japanese', 39),
 (u'indian', 29),
 (u'italian', 26),
 (u'thai', 25),
 (u'american', 23),
 (u'sushi', 19)]
```

SUGGESTED IMPROVEMENTS

There are a few areas of improvements that would lead to better analysis

(1) Incompleteness of Data:

- The data seems to be incomplete in several areas that will help in association and analysis.
- There are several standalone pieces of data that are not well connected. For example, as seen in the snippets of code given below, there are over a 100 beauty shops but only 4 of them have associated city information.

In [ ]:
```
cur.execute("SELECT value, count(*)  \
            FROM nodes_tags \
            WHERE key = 'shop' GROUP BY 1 ORDER BY 2 DESC LIMIT 5 ")
pprint.pprint(cur.fetchall())
```

In [ ]:
```
[(u'beauty', 104),
 (u'clothes', 100),
 (u'supermarket', 99),
 (u'hairdresser', 98),
 (u'car_repair', 82)]
```

In [ ]:
```
cur.execute("SELECT  nt1.value, count(*) \
            FROM nodes_tags nt1 \
            JOIN \
                (SELECT distinct(id)  \
                FROM nodes_tags \
                WHERE key = 'shop' and value = 'beauty') nt2 \
            ON nt1.id = nt2.id \
            WHERE nt1.type = 'addr' and nt1.key = 'city' \
            GROUP BY 1 ORDER BY 2 DESC ")
pprint.pprint(cur.fetchall())
```

In [ ]:
```
[(u'San Jose', 1), (u'San Jos\xe9', 1), (u'Santa Clara', 1), (u'Sunnyv
ale', 1)]
```

- This is understandable as this is user-added data but it is also something that will be gradually addressed over time.
- One potential source of immediate improvement is to use other sources of data as well. Each city must be having available resources that can be tapped into.

(2) Data Formatting issues:

- There are formatting issues in some user-inputted fields like restaurant names and also city names as seen above. Also, phone numbers seem to be inconsistently formatted.
- These can be remedied by adding user-data checks or a input parser that removes such encoded data and/or updates using built-in regexp based mappings.
- We may run into issues due to language and scripts like Chinese or Devanagari (Hindi). We could look into solutions like having a UTF-8 character regex-based mappings for various symbols and languages. These will take longer to implement but will be useful to more and more people.

CONCLUSION

This project was a great learning experience in finding out more about the place where I live and also in understanding and processing OSM data.