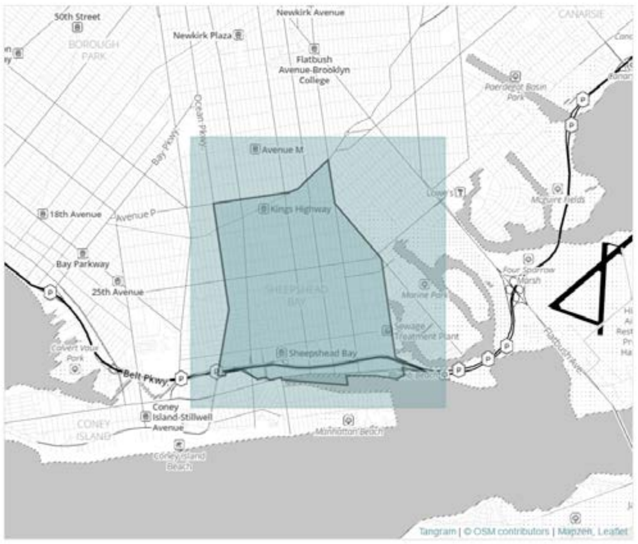
OpenStreetMap Data Case Study

For this project, I chose to study my childhood neighborhood: Sheepshead Bay, Brooklyn, NY.



Problems Encountered

I encountered the following problems or anomalies in the .osm file:

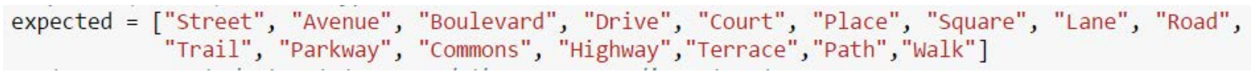
- Overabbreviated street names & extra street types
- Inconsistent zip codes
- Extra “k” level tags (Tiger data, cityrack data, gnis data)

Street Names & Types

The street names were much more consistent than I had anticipated, compared to the Chicago example in the Udacity course. Initially, there was only one overabbreviation, but that number ballooned to about 10 when I accounted for data in other “k” formats.

```
defaultdict(<type 'set'>, {'North': set(['Shore Parkway Sr North', 'Village Road North']), '218650358': set(['218650358']), 'Pla
za': set(['Kings Plaza']), 'St': set(['Kimball St', 'E 17th St', 'Knapp St']), 'Rd': set(['Quentin Rd']), 'Av': set(['Coney Isla
nd Av', 'Ocean Av']), 'Ave': set(['Ave N', 'Ave S', 'Ave X', 'Flatbush Ave', 'Emmons Ave', 'Bedford Ave', 'Ave U', 'Gerritsen Av
e', 'Nostrand Ave']), 'East': set(['Village Road East']), 'avenue': set(['Bedford avenue']), 'South': set(['Shore Parkway Sr Sou
th', 'Village Road South'])})
```

In addition to the street types from the exercise, I also added “Highway”, “Terrace”, “Path”, and “Walk.”



```
expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons", "Highway","Terrace","Path","Walk"]
```

Other streets ended in a cardinal direction, such as “Village Road North,” which did not needed to be changed. There was also one abbreviation, “Sr” in “Shore Parkway Sr North”, but I could not figure out what it was short for, so I left it as is.

There was one street that was all numbers. It is skipped when checking whether the tag has special characters.

Lastly, there were some streets that started with the street types, such as “Avenue T.” Some of these had overabbreviations, so I had to extract the first part of the string and update those as well.

Here is the street audit output:

```
defaultdict(<type 'set'>, {'North': set(['Shore Parkway Sr North', 'Village Road North']), '218650358': set(['218650358']), 'Pla
za': set(['Kings Plaza']), 'St': set(['Kimball St', 'E 17th St', 'Knapp St']), 'Rd': set(['Quentin Rd']), 'Av': set(['Coney Isla
nd Av', 'Ocean Av']), 'Ave': set(['Ave N', 'Ave S', 'Ave X', 'Flatbush Ave', 'Emmons Ave', 'Bedford Ave', 'Ave U', 'Gerritsen Av
e', 'Nostrand Ave']), 'East': set(['Village Road East']), 'avenue': set(['Bedford avenue']), 'South': set(['Shore Parkway Sr Sou
th', 'Village Road South'])})
```

Zip Codes

The zip codes were all accurate and mostly consistent. There were four cases where the zip codes included 9 digits, so I simply updated them by truncating it to 5 characters.

Here is the zip code audit output:

```
{'11223': 5071, '11224-4003': 3, '11210': 1257, '11229': 12347, '11230': 3402, '11224': 140, '11234': 7067, '11235': 8082}
```

Extra Data Sources

The data included sources that required special attention to deal with. These were tags that had “k” values from tiger, gnis and cityracks.

After auditing the data, the following changes were necessary:

- tiger values were being handled correctly since they use semicolons, but they included zip codes using “tiger:zip_left” and “tiger:zip_right.” When checking tags if they were zip codes, I included these as well:
```
def is_zip(elem):
    return elem.get('k') == 'addr:postcode' or elem.get('k') == 'tiger:zip_left' or elem.get('k') == 'tiger:zip_right'
```
- gnis data were handled correctly as well, but gnis values did not actually include anything of interest. Things like street names and zip codes were included separately, so I did not have to extract any data from it.
- cityracks data were used for bicycle parking in the area. It used a period, so I needed to make sure I split it correctly. Further, streets were included using “cityracks.street,” so I had to handle that as such:
```
def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street" or elem.attrib['k'] == "cityracks.street")
```

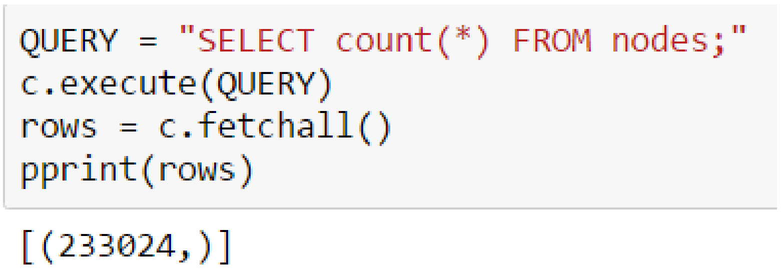
Data Overview

File Sizes

File	Size
Sheepshead Bay.osm	61 MB
Sheepshead.db	36.6 MB
nodes.csv	21.5 MB
nodes_tags.csv	575 KB
ways.csv	3.2 MB
ways_nodes.csv	7.8 MB
ways_tags.csv	8.2 MB

Number of Nodes

There are 233,024 nodes in the data.

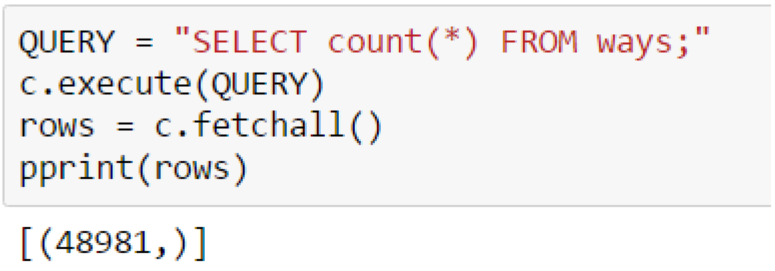


```
QUERY = "SELECT count(*) FROM nodes;"
c.execute(QUERY)
rows = c.fetchall()
pprint(rows)
```

```
[(233024,)]
```

Number of Ways

There are 48981 ways in the data.



```
QUERY = "SELECT count(*) FROM ways;"
c.execute(QUERY)
rows = c.fetchall()
pprint(rows)
```

```
[(48981,)]
```

Number of Unique Users

There are 129 unique users in the data.

```
QUERY = "SELECT count(DISTINCT(uid)) from (select uid FROM ways union all select DISTINCT uid from nodes);"
c.execute(QUERY)
users = c.fetchall()
pprint(users)
```

```
[(129,)]
```

```
QUERY = '''SELECT user, COUNT(*) as num
FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways)
GROUP BY user
ORDER BY num DESC
LIMIT 10;'''
c.execute(QUERY)
topusers = c.fetchall()
pprint(topusers)
```

```
[(u'Rub21_nycbuildings', 244623),
 (u'ingalls', 15758),
 (u'celosia_nycbuildings', 4501),
 (u'ingalls_nycbuildings', 3339),
 (u'aaron_nycbuildings', 2434),
 (u'watame', 1782),
 (u'ediyes_nycbuildings', 1552),
 (u'rjhale1971', 1255),
 (u'woodpeck_fixbot', 1011),
 (u'MxxCon', 884)]
```

Additional Statistics & Data Exploration

Using SQL, additional statistics can be calculated. Two problems were encountered when using SQL to explore the csv tables:

- There were some foreign characters that SQL had trouble handling. As a result, the csv files has to be parsed and converted to SQL tables in python, and the “decode("utf-8")” function was used.
- A number of the explorations required pulling tags from both nodes_tags and way_tags. The ids on these were different, so an outer join was needed. However, the full outer join could not be implemented in sqlite3, so a UNION ALL was used (or in some cases, a very creative join).

The explorations and statistics are shown below.

Top 10 Amenities

```
#Amenities
QUERY = '''SELECT value, COUNT(*) as num
FROM (SELECT value,key FROM nodes_tags UNION ALL SELECT value,key FROM ways_tags)
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;'''
c.execute(QUERY)
amenity = c.fetchall()
pprint(amenity)
```

```
[(u'parking', 135),
 (u'school', 63),
 (u'bench', 59),
 (u'place_of_worship', 57),
 (u'bicycle_parking', 34),
 (u'bank', 14),
 (u'fire_station', 11),
 (u'fuel', 10),
 (u'pharmacy', 10),
 (u'restaurant', 7)]
```

Top Religion (By Place of Worship)

```
#Top religion for place of worship
QUERY = '''SELECT l.value, l.count1 +l.count2 as num
from ((SELECT nodes_tags.value, COUNT(*) as count1
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') as i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value) as j
JOIN (SELECT ways_tags.value, COUNT(*) as count2
FROM ways_tags
    JOIN (SELECT DISTINCT(id) FROM ways_tags WHERE value='place_of_worship') as k
    ON ways_tags.id=k.id
WHERE ways_tags.key='religion'
GROUP BY ways_tags.value) as m
on j.value = m.value) as l
group by l.value
order by num desc
limit 1;
'''
c.execute(QUERY)
religion = c.fetchall()
pprint(religion)
```

```
[(u'jewish', 35)]
```

Top 10 Streets with Most Data

```
#Streets with most data
QUERY = '''SELECT value, COUNT(*) as num
FROM (SELECT value,key FROM nodes_tags UNION ALL SELECT value,key FROM ways_tags)
WHERE key='street'
GROUP BY value
ORDER BY num DESC
LIMIT 10;'''
c.execute(QUERY)
street = c.fetchall()
pprint(street)
```

```
[(u'East 28th Street', 755),
 (u'East 26th Street', 751),
 (u'East 29th Street', 715),
 (u'Bedford Avenue', 674),
 (u'East 27th Street', 648),
 (u'East 23rd Street', 626),
 (u'East 21st Street', 612),
 (u'East 22nd Street', 600),
 (u'Coney Island Avenue', 593),
 (u'East 15th Street', 585)]
```

Note: Bedford Avenue is equivalent to East 25th Street and Coney Island Avenue is equivalent to East 10th Street. It seems that the streets with most populated data are all parallel to one another, running north to south!

Average Height of Building

```
#Average height of building
QUERY = '''SELECT (sum(value)/COUNT(*)) as num
FROM (SELECT value,key FROM nodes_tags UNION ALL SELECT value,key FROM ways_tags)
WHERE key='height';'''
c.execute(QUERY)
height = c.fetchall()
pprint(height)
```

```
[(7.240978514108155,)]
```

According to the OSM wiki, the default unit of length if not specified is meters (http://wiki.openstreetmap.org/wiki/Map_Features/Units). Thus, the average height of buildings in this area is about 7.2 meters (about 4 floors!).

Conclusion

After exploring and reviewing the data, a couple of things struck me. First, I did not expect my neighborhood to have this much data! The file size for just my own neighborhood was over 60MB. Second, the user data was much cleaner than I had expected; before including tiger, cityrack and gnis data, the street names and zip codes were pretty consistent and the data included a lot of detail for the ways and nodes that were present. Lastly, the data is very incomplete. For example, there were only 7 restaurants in the entire dataset, but I can see 7 restaurants just looking out my parents' apartment's window. If it were to thoroughly include more data, there is definitely a lot of potential with the OSM platform, especially the inclusion of the location of public toilets (!!), which can be quite the commodity in NYC!

An improvement for the data itself would be supplementing the data from other sources. Since it seems like building and street information is present, but not actual business information, a database of businesses and shops would be a good supplement. Yelp includes details such as hours and business types and Google Maps includes latitude and longitude information, so they would be a good combination of sources to pull from. The data seems to be readily available, so it should be possible to create a script to pull the data into OSM. However, some difficulties may be if the information from the two sources do not match up. It would also be pretty redundant to make OSM a copy of Google Maps; the great part about OSM is that it includes data, such as parking, benches and toilets, that are not in Google Maps, so other data sources would be needed as well.

As for the data cleaning and exploring process itself, the painstaking process of cleaning demonstrated very clearly to me the importance of making sure the data quality is high in the first place. Still, python + SQL is a very combination of tools that can quickly remedy a bad dataset. I also found the SQL queries easy to understand, yet surprisingly complex, such as when doing things like self joins and subqueries or working around the absence of the full outer join function.

For analyzing the data, it would have been very helpful if the ways_tags and nodes_tags were in one csv/table. That way, querying tags wouldn't require a join. However, it would be more difficult and slower if a user were to only want data from one source, such as if they wanted to do a join between nodes and nodes_tags.