NEW YORK

I decided to wrangle the open street map data of the great city of New York. New York is currently the city I have lived for more than 10 years and have familiarity with much of the city.
Source: https://mapzen.com/data/metro-extracts/metro/new-york_new-york/ (https://mapzen.com/data/metro-extracts/metro/new-york_new-york/)

Problems with the Data

About 2,800 different users input data into the open street map sample for New York, NY. One would expect with this many different users performing data entry that there would be some issues with consistency and errors with data entered. I specifically looked issues with address and attempted to rectify them programmatically. The issues I found and addressed specifically were: 1.) Varying names in street addresses including abbreviations like St, and Ave. and also full names like Street and Address. 2.) A few misspellings in the street address. 3.) Varying names for State including popular abbreviations and states that do not belong in the data set. 4.) Zipcodes that inconsistently included the 4 additional postal digits.

Wrangling the street Names

I observed several instances where the name of street needed cleaning to make the street address more uniform. There were issues with abbreviations and also inconsistent and incorrect capitalization. I created a program to iterate through all of the raw street data and replace any street name that was abbreviated or had a capitalization issue. I also insured that any street name was spelled correctly.

An example of the issue I found with the data: {'Ave':'Franklin Ave', 'N Passaic Ave', 'Anderson Ave', '4th Ave', 'Millard Ave', 'Inman Ave', 'Bloomfied Ave', 'Myrtle Ave', 'Flatbush Ave', 'Willow Ave', 'Plainfield Ave', 'W Allendale Ave', '2nd Ave', 'Lakeview Ave', 'Van Houten Ave', 'Park Ave', 'Cypress Ave'}

In [2]:
```
# Clean Street Types.
#Mapping of street names observed in the list of data of unexpected street types.
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

street_mapping = { "St": "Street",
            "St.": "Street",
            "STREET":"Street",
            "Rd": "Road",
            "Rd.": "Road",
            "Ave": "Avenue",
            "Ave.":"Avenue",
            "avenue":"Avenue",
           "AVENUE":"Avenue",
           "Avene":"Avenue",
            "S":"South",
           "W":"West",
           "N":"North",
           "E":"East",
           "Pkwy":"Parkway",
           "Cir":"Circle",
           "Blvd":"Boulevard"
            }

#Function to take an unexpected name and replace it with a new name from mapping.
def update_street_name(name, mapping):
    m = street_type_re.search(name)
    if m:
        street_type = m.group()
        if street_type in mapping:
            street_type_fixed=mapping.get(street_type,)
            name = name.replace(street_type, street_type_fixed)
    return name
```

Postal Code/Zip Code

The next information that I reviewed was the postal code also known as the zip code. Postal code can easily vary due to 9 digit postal codes as well as computational errors. Most but not all NYC postal codes start with X. Since New York City has such a vast area and is so densly populated there are many more postal codes than other cities.

Initially I audited a small subsection of the postal code. I realized that I would need to cross check the zipcodes in order to properly wrangle the data. I found an xml file of zipcodes on the NYC department of health's website. I sorted through that data to come up with a list of zipcodes for NYC. I was surprised to learn that NYC has 236 zipcodes.

In [2]:
```
zipcode_file='rows.xml'

def create_list_zipcode(file): #takes a file that contains zipcodes and returns al
l of the zipcodes.
    mapping=[]
    for _, element in ET.iterparse(file):
        for child in element:
            if child.tag=="jurisdiction_name":
                mapping.append(child.text)
    return mapping
```

In order to audit the zip codes I first verified it had the correct amount of digits. A US zip code typically contains five digits but can contain an additional four digits (used by the postal service). I assumed that it was possible that the open street map field "postal code" contained nine digits as well as other errors. Additionally, I verified that any five digit zip code was a NYC zip code.

In [ ]:
```
zip_code_re= re.compile('\d{5}$', re.IGNORECASE)


zipcodes_expected=create_list_zipcode(zipcode_file)

def audit_street_type(zipcode_types, zip_code):
    m = zip_code_re.search(zip_code)
    if m:
        zip_type = m.group()
        if zip_code not in zipcodes_expected:
            zipcode_types['Not NYC Zip']=zip_type
    else:
        zipcode_types[m].add(zip_code)

def is_zipcode_name(elem):
    return (elem.attrib['k'] == "addr:postcode")

def update_zipcode_name(name, mapping):
    m = zip_code_re.search(name)
    if m:
        zipcode = m.group()
        if zipcode in mapping:
            name = name.replace(zipcode, name)
    return name

def audit(osmfile):
    osm_file = open(osmfile, "r")
    zipcode_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):

                if is_zipcode_name(tag):

                    audit_street_type(zipcode_types, tag.attrib['v'])

    osm_file.close()
    return zipcode_types
```

Defaultdict(<class 'set'>, {'Not NYC Zip': '11735', None: {'08901-1400', '08901-1108', '08901-8525', '08901-1004', '08854-8019', '08854-3929', '08854-8006', '08901-8545', '08901-8557', '08901-1111', '08854-8009', '08901-8542', '08901-2882', '08901-2878', '08901-1166', '08901-2831', '08854-8035', '11374-2756', '08854-8000', '08901-2832', '08854-8063', '08901-8548', '07501-210', '08901-8529', '08854-8010', '08854-8032', '08854-8023', '08854-8052', '08901-1904', '08901-8519', '08901-8504', '08901-1282', '10017-6927', '08854-8045', '83', '08901-1176', '08854-8012', '08901-2867', '08901-8502', '08901-8558', '08901-8513', '08901-8528', '08854-8014'}})

To fix the 9 digit zipcode list I decided to write code that utilized the regular expression to take the first five digits.

def update_state(name, mapping): m = state_type_re.search(name) if not m: state_type = name if state_type in mapping: state_type_fixed=mapping.get(name,) name = name.replace(state_type, state_type_fixed) return name

From the SQL database I double checked that the data had been properly cleaned.

In [ ]:
```
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
UNION ALL
SELECT * FROM ways_tags) tags
WHERE tags.key='postcode'
GROUP BY tags.value
ORDER BY count DESC;
```

While the cleaning was able to remove the 9 digit zipcode there were still several items that needed further individual review and verifification. Including: 83, NY 10, 12692, 22645. As you can see there are items that are complete error entries but also zipcodes that are not from the New York region.

Then I pulled the top 10 zip codes from the sample and verified they are in the NYC area. The top zipcodes were not in the city center but were within the tristate area.

SELECT tags.value, COUNT*() as count FROM (SELECT* FROM nodes_tags UNION ALL SELECT * FROM ways_tags) tags WHERE tags.key='postcode' GROUP BY tags.value ORDER BY count DESC LIMIT 10;

Zipcode Count
10314|2296 11234|1984 10312|1781 10306|1607 11236|1528 11746|1501 11385|1500 11706|1472 11743|1401 11757|1303

The last aspect of the code that I decided to be wrangled was the state in the address. New York City and it's suburbs are in three states: New York, New Jersey, and Conneticut. Due to the vast area and also the different naming conventions and abbreviations that could be used for the states.

In [ ]:
```
state_type_re = re.compile('qr/(Alabama|Alaska|Arizona|Arkansas|California|Colorad
o|Connecticut|Delaware|Florida|Georgia|Hawaii|Idaho|Illinois|Indiana|Iowa|Kansas|K
entucky|Louisiana|Maine|Maryland|Massachusetts|Michigan|Minnesota|Mississippi|Miss
ouri|Montana|Nebraska|Nevada|New\sHampshire|New\sJersey|New\sMexico|New\sYork|Nort
h\sCarolina|North\sDakota|Ohio|Oklahoma|Oregon|Pennsylvania|Rhode\sIsland|South\sC
arolina|South\sDakota|Tennessee|Texas|Utah|Vermont|Virginia|Washington|West\sVirgi
nia|Wisconsin|Wyoming)/', re.IGNORECASE)


expected = ["New York", "New Jersey","Conneticut"]


def audit_state(states, state_name):
    m = street_type_re.search(state_name)
    if m:
        if state_name not in expected:
            states[state_name].add(state_name)


def is_state_addr(elem):
    return (elem.attrib['k'] == "addr:state")

# Changes Street types
state_mapping = { "NY": "New York",
            "ny": "New York",
           "Ny": "New York",
           "NJ.": "New Jersey",
            "Nj":"Ner Jersey",
           "nj": "Ner Jersey"
            }


def update_state(name, mapping):
    m = state_type_re.search(name)
    if not m:
        state_type = name
        if state_type in mapping:
            state_type_fixed=mapping.get(name,)
            name = name.replace(state_type, state_type_fixed)
    return name


def audit(osmfile):
    osm_file = open(osmfile, "r")
    states = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):

                if is_state_addr(tag):

                    audit_state(states, tag.attrib['v'])

    osm_file.close()
    return states
```

As expected I found several different abbreviation variations in the state field. For example New York was featured as ny, NY, N.Y., and Ny. All of this type of variation can be fixed programatically with an update name function.

As a part of my review I pulled a sample of some of the keys.

SELECT key, value, type FROM nodes_tags GROUP BY value ORDER BY key DESC;

I verified the state and noticed that while the changes that were made utilzing the code and found the followng:

state|Conneticut|addr state|New Jersey|addr state|New York|addr state|VA|addr

The record or records that create the VA address will need to be looked at individuallly as Virginia is not in the NYC metro area and the record must have an error.

In [ ]:
```
More information about a DataSet:
```

In [ ]:
```
The file size was as follows:
    Original OSM - new-york_new-york.osm 2.83 GB
    Sample OSM - sample.osm 286.7 MB
    Database - new2.db 207.8 MB
```

Utilizing the request below in the sample nyc data utilzed there were 181,560 ways and 1,157,526 nodes.

SELECT COUNT*() as num FROM ways; and SELECT COUNT()* as num FROM nodes;

In [ ]:
```
I then looked into the Unique Users for the data set.

SELECT COUNT(*) FROM
(SELECT DISTINCT nodes.uid FROM nodes UNION SELECT DISTINCT ways.uid FROM ways);

In the sample there were 2,779 unique users.
```

Then used the following function to look up the top 15 users that posted nodes and ways: SELECT user, COUNT(*) as num FROM (SELECT nodes.user FROM nodes UNION ALL SELECT ways.user FROM ways) GROUP BY user ORDER BY num DESC LIMIT 15;A breakdown of the top users: User | Count of Posts Rub21_nycbuildings|488428 ingalls_nycbuildings|93577 MySuffolkNY|62643 woodpeck_fixbot|61145 SuffolkNY|58037 minewman|49400 Northfork|41342 ediyes_nycbuildings|27196 lxbarth_nycbuildings|23289 smlevine|21976 NJDataUploads|19714 ingalls|19092 JoelManagua|18705 CoreyFarwell|17085 RI-Improve|15350

In [ ]:
```
I then decided to look at the ammenities offered.
```

In [ ]:
```
SELECT key, value, COUNT(*) as num FROM nodes_tags
WHERE key="amenity"
GROUP BY value
ORDER BY num DESC
LIMIT 15;
```

amenity|bicycle_parking|526 amenity|school|337 amenity|restaurant|328 amenity|place_of_worship|302 amenity|cafe|111 amenity|fast_food|102 amenity|bench|75 amenity|fire_station|58 amenity|bank|55 amenity|drinking_water|54 amenity|bar|51 amenity|pharmacy|50 amenity|post_box|46 amenity|bicycle_rental|42 amenity|toilets|36

It's interesting that bicycle parking was the largest ammenity in the sample. In recent years an emphasis on bike lanes and bicycles has been a large area of interest in the city. With the introduction of citibike as well as the increasing popularity of bicycles it would make sense that many many parking spaces for bicycles would be widely available in the city.

Another thing I noted and to be honest was loooking for is a large amount of food establishments, if you add restuarant, cafe, and fast food together (I would consider all a type of eating establishment) you have 541 places. Again NYC is a place known for having lots of places to eat so this makes so much sense.

THe next key I decided to look into was buildings. NYC offers a variety of building type sand sizes.

In [ ]:
```
SELECT e.key, e.value, COUNT(*) as num
FROM (SELECT * FROM nodes_tags
UNION ALL
SELECT * FROM ways_tags) e
WHERE e.key="building"
GROUP BY e.value
ORDER BY num DESC;
```

In [ ]:
```
A snap shot of the sample:
key     value  count
building|yes|113062
building|house|16256
building|garage|12266
building|shed|1591
building|commercial|951
building|service|712
building|apartments|324
building|residential|285
building|school|151
building|retail|147
building|industrial|134
```

Additional wrangling will need to be done with the data to identify what is "yes" is in terms of building and what values should these items should be changed to. It's also interesting that house and garage are the largest portion of the data other than "yes". It's possible that building was meant to indicate that it was a larger dwelling other than a house but we can not be certain without a longer review of the data.

Conclusion

It is obvious that the data requires a bit more wrangling and also some manual interventions to fix one off inaccuracies. While the open street map offers a wealth of information about the world there are a variety of issues within the dataset that prevent it from being the most accurate source of information. One suggestion to further improve the open street map data would be to implement a drop down box for the state and country data. To take this idea even further, implementing an authentication tool that autofilled or correct the complete street address when inputted for example:930 53rd st ny, ny 10567-9876 (fake address) would automatically be changed to 930 53rd Street New York, New York 10567 (fake address). This type of authentication could eliminate some of the consistency issues with the street address and would make the data more uniform. On the other hand, an authenticaion tool may be costly to implement and maintain. Also authentication would only fix any new entries into open street map and would not address many of the noted issues. Another solution would still be required to fix the existing data. As machine learning technology improves and becomes more ubiquitous in our daily lives an inexpensive solution will be created and machine intervention will fix much of the data in Open Street Map. Open Street Map as is, is still an excellent tool and provides anyone a wealth of information about cities all over the world.

In [ ]:
```
```