OpenStreetMap data Wrangling Project

I will leverage this notebook to summarize my steps and findings for the Data Wrangling Project, using the data downloaded from OpenStreetMap.

Introduction

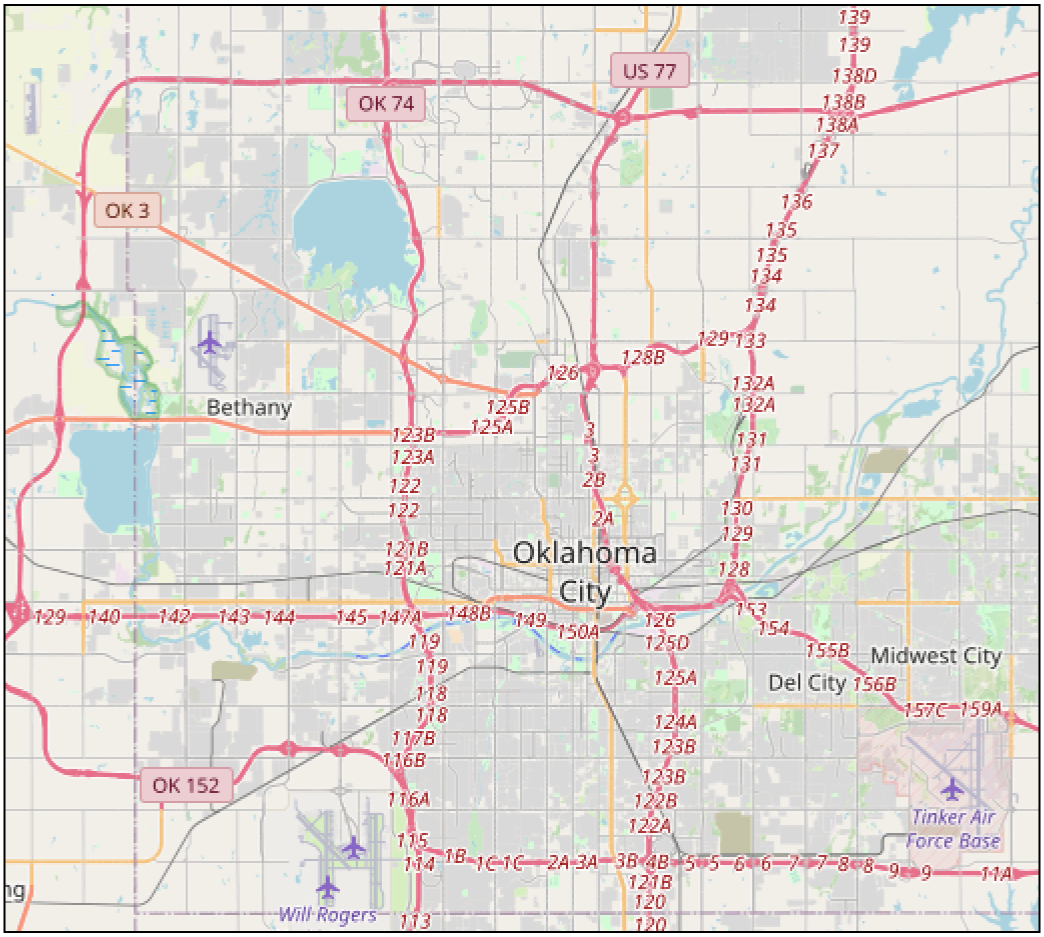
For my project, I decided to analyze the OpenStreetMap data for Oklahoma City, Oklahoma. I grew up near this city, went to college at the local university, and lived in the area for several years before moving to Houston Texas. For the analysis, I went with SQL for the data wrangling.

OpenStreetMap Link - https://www.openstreetmap.org/export#map=12/35.5090/-97.6334

In [3]:

```
from IPython.display import Image
Image("Oklahoma_City.png")
```

Out[3]:



Objectives

- Assess the overall quality of the data:
  - Validity
  - Accuracy
  - Completeness
  - Consistency
  - Uniformity
- Be able to parse and gather the data
- Be able to process the data
- Learn how to:
  - Store
  - Query
  - Aggregate data with SQL

Data Auditing and Cleaning

- I leveraged identifyTags.py to help identify the tags used in the datafile.
  - I will be looking for the Node and Way tags.
- TagAudit.py was used to look for:
  - Tags with only lowercase letters
  - Tages with lowercase letters separated by a colon

  - Search for any problem characters
- AuditingK.py was used to find the different attributes represented by the 'k' value and measure their occurrence.
- I used StreetTypes.py to audit the street names in the yukon.osm file.
- UpdateStreetTypes.py was used to correct inconsistencies with street names in the yukon.osm file.

Map Impoer and Tag Identifucation

This function will display the types of elements found in the OSM file, and help us determine which ones are important. I went with Overpass_API to import the map of Oklahoma City. Using Overpass-API returned an .osm file of the area.

In [1]:

```
import xml.etree.cElementTree as ET
import pprint

OSMFILE = 'Oklahoma_City.osm'
SAMPLE_FILE = 'sample.osm'

def count_tags(filename):
    tags = {} #create empty dic to hold values of tags and their counts
    for event, elem in ET.iterparse(filename):
        if elem.tag not in tags.keys():
            tags[elem.tag] = 1
        else:
            tags[elem.tag] += 1
    return tags

tags = count_tags(OSMFILE)
pprint.pprint(tags)
```

```
{'bounds': 1,
 'member': 49902,
 'nd': 2877209,
 'node': 2476566,
 'osm': 1,
 'relation': 4475,
 'tag': 1311072,
 'way': 263981}
```

Now that we have the elements, we can recognize the type of top level tags that are given in the OpenStreetMap Wiki.

Next, we will review the size of the dataset we will be working with throughout this exercise.

Size of the original file

In [2]:

```
import os
bytes = os.path.getsize('Oklahoma_City.osm')
mb = float(bytes / 1000000)
print ("osm file size:", mb, "Mb")
```

```
osm file size: 555.209474 Mb
```

We have a large dataset to work with. Lets create a smaller sample set.

Sample file creation

The following function creates a sample file. (see sample.jpynb)

In [4]:

```
k = 25 # Parameter: take every k-th top level element. The value was tuned to get the a

def get_element(filename, tags=('node', 'way', 'relation')):
    context = iter(ET.iterparse(filename, events=('start', 'end')))
    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()

with open(SAMPLE_FILE, 'w',encoding='utf-8') as output:
    output.write('<?xml version="1.0" encoding="UTF-8"?>\n')
    output.write('<osm>\n  ')
    # Write every kth top level element
    for i, element in enumerate(get_element(OSMFILE)):
        if i % k == 0:
            output.write(ET.tostring(element, encoding='utf-8').decode())
    output.write('</osm>')
```

Now we will check to see if the file was created and find out the size of the sample.

In [5]:

```
import os
bytes = os.path.getsize('sample.osm')
mb = float(bytes / 1000000)
print ("osm file size:", mb, "Mb")
```

```
osm file size: 22.70869 Mb
```

The sample file is much smaller than the original dataset.

Auditing the "k" values

We will use the following function to find the attributes that are represented by the "k" value, and use the results to measure the number of occurrences in the dataset. (AuditingK.jpynb)

In [7]:

```
import pprint
import xml.etree.cElementTree as ET

def get_types_of_k_attrib(filename, k_attrib_values_dict):

    for _, element in ET.iterparse(filename):
        if element.tag == "node" or element.tag == "way":
            for tag in element.iter("tag"):
                #print(tag.attrib['k'])
                if tag.attrib['k'] not in k_attrib_values_dict:
                    k_attrib_values_dict[tag.attrib['k']] = 1
                else:
                    k_attrib_values_dict[tag.attrib['k']] += 1
                tag.clear()
            element.clear()

if __name__ == '__main__':

    k_attrib_values_dict = {}
    filename = "Oklahoma_City.osm"
    get_types_of_k_attrib(filename, k_attrib_values_dict)

    #print the top 10 k values appearing in the Oklahoma_City.osm file
    import operator
    pprint.pprint(sorted(k_attrib_values_dict.items(),key = operator.itemgetter(1),reve
```

```
[('name', 61869),
 ('source', 59094),
 ('building', 51525),
 ('tiger:county', 47465),
 ('tiger:cfcc', 47005),
 ('service', 38395),
 ('tiger:name_base', 38119),
 ('access', 36282),
 ('tiger:name_type', 35649),
 ('tiger:zip_left', 32514)]
```

Problem Search

Our next function will look for tags that contain only lowercase letters, after that we will look for lowercase letters separated by a colon(:), then we will try to identify problem characters within the dataset. (TagAudit.jpynb)

In [8]:

```
import xml.etree.cElementTree as ET
from collections import defaultdict
import pprint
import re
import csv
import codecs
#import schema
import sqlite3
import pandas as pd

lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&;\'"\?%#$@\,\. \t\r\n]')


def key_type(element, keys):
    if element.tag == "tag":
            if lower.search(element.attrib['k']) != None:
                keys['lower'] += 1
            elif lower_colon.search(element.attrib['k']) != None:
                keys['lower_colon'] += 1
            elif problemchars.search(element.attrib['k']) != None:
                keys['problemchars'] += 1
            else:
                keys['other'] += 1


    return keys

def process_map(file):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(file):
        keys = key_type(element, keys)
```

```python
    return keys
process_map('Oklahoma_City.osm')
```

Out[8]:

```
{'lower': 709610, 'lower_colon': 396310, 'problemchars': 0, 'other': 205152}
```

Map Contributors

Lets try to determine the number of contributors to our dataset. First we will get the total number of contributors and then we will find the top ten.

In [10]:

```python
def get_user(element):
    return

def process_map_user(file):
    users = set()
    for _, element in ET.iterparse(file):
        if element.tag == 'node' or element.tag == 'way' or element.tag == 'relation':
            users.add(element.attrib['user'])
    print("Total Contributors To This Map Area is:", len(users))
process_map_user('Oklahoma_City.osm')
```

```
Total Contributors To This Map Area is: 1519
```

Our dataset has 1,519 contributors. Now let's find the top ten.

I will need to bring in some new libraries before completing this step.

In [12]:

```python
# importing all necessary libraries
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

def execute_query(QUERY, header):
    db = sqlite3.connect("Oklahoma_City.db")
    c = db.cursor()
    c.execute(QUERY)
    rows = c.fetchall()
    df = pd.DataFrame(rows, columns = header)
    db.close()
    return df

# Style and size of graph
sns.set_style('darkgrid')
sns.set(rc={'figure.figsize':(8,6)})
QUERY = "SELECT user, COUNT(*) AS number \
FROM (SELECT user, uid FROM nodes UNION ALL SELECT user, uid FROM ways) \
GROUP BY uid \
ORDER BY number DESC \
LIMIT 10;"

execute_query(QUERY, ['user', 'number'])
```

Out[12]:

	user	number
0	OklaNHD	766976
1	JamesTheElder	185540

	user	number
2	Dulahey	177783
3	Baloo Uriza	174581
4	dufekin	160681
5	Fluous	121958
6	dustybrimaps	115197
7	William McBroom	65941
8	woodpeck_fixbot	56719
9	SathyaPendyala	52949

Street Type Auditing

We will start with a list of the expected values, them map varisions to their expected value

In [13]:

```
expected = ['Street', 'Avenue', 'Boulevard', 'Road', 'Place', 'Parkway', 'Lane',
            'Drive']
```

In [14]:

```
mapping = { "St": "Street",
            "St.": "Street",
            "street": "Street",
            "Ave": "Avenue",
            "Ave.": "Avenue",
            "Blvd": "Boulevard",
            "Blvd.": "Boulevard",
            "Boulavard": "Boulevard",
            "Rd": "Road",
            "Rd.": "Road",
            "RD": "Road",
            "Pl": "Place",
            "Pl.": "Place",
            "PKWY": "Parkway",
            "Pkwy": "Parkway",
            "Ln": "Lane",
            "Ln.": "Lane",
            "Dr": "Drive",
            "Dr.": "Drive"
            }
```

In [15]:

```
import xml.etree.cElementTree as ET
import pprint
import re
from collections import defaultdict

datadir = "data"
datafile = "Oklahoma_City.osm"

street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

expected = ['Street', 'Avenue', 'Boulevard', 'Road', 'Place', 'Parkway', 'Lane',
            'Drive']
```

```
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)


def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")


def audit(osmfile):
    osm_file = open(osmfile, "r", errors = 'ignore')
    street_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_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    osm_file.close()
    return street_types

Oklahoma_city_street_types = audit(datafile)
pprint.pprint(dict(Oklahoma_city_street_types))
```

```
{'1017': {'Northwest Expressway Ste 1017'},
 '103': {'N Classen Blvd #103'},
 '150th': {'NW 150th'},
 '152': {'East State Highway 152'},
 '2000': {'North Robinson Avenue Suite 2000'},
 '23rd': {'NW 23rd', 'NEW 23rd'},
 '3357': {'South 3357'},
 '37': {'State Highway 37'},
 '66': {'Highway 66', 'East Highway 66'},
 '73069': {'73069'},
 '76': {'South Highway 76'},
 '937': {'Terminal Drive, Unit 937'},
 'AVE': {'NORTH PENNSYLVANIA AVE', 'NORTH MAY AVE'},
 'Ave': {'7700 North Hudson Ave',
         'N Meridian Ave',
         'N Western Ave',
         'North May Ave',
         'S Pennsylvania Ave',
         'S Rock Island Ave',
         'S Western Ave',
         'S Yukon Ave',
         'S. Meridian Ave',
         'South Portland Ave',
         'South Robinson Ave',
         'W. Sheridan Ave'},
 'Ave.': {'South Walker Ave.', 'W. Sheridan Ave.', 'East California Ave.'},
 'Blvd': {'Garth Brooks Blvd',
          'Highland Park Blvd',
          'N MacArthur Blvd',
          'N Midwest Blvd',
          'S Douglas Blvd',
          'William Penn Blvd'},
 'Broadway': {'South Broadway', 'N. Broadway'},
 'Circle': {'Bradford Circle',
            'Foxwood Circle',
            'Halley Circle',
            'Maehs Circle',
            'North Classen Circle',
            'North Geary Circle',
            'Northwest 165th Circle',
            'Pinyon Circle',
            'Replublic Circle',
            'Republic Avenue Circle',
            'Scirocco Circle',
            'Tioga Circle'},
 'Court': {'Broce Court',
           'Classen Court',
           'Fox Bluff Court',
           'Glendover Court',
           'Halifax Court',
           'Montane Court',
           'Planet Court'},
 'Crossing': {'Lanes Crossing'},
 'DR': {'Riverwalk DR'},
 'Danforth': {'West Danforth'},
 'Diagonal': {'Tinker Diagonal'},
 'Dr': {'Outlet Shops Dr', 'Babcock Dr', 'Cornwell Dr'},
 'Dr.': {'6604 Willowridge Dr.', 'Avondale Dr.', 'West Lake Hefner Dr.'},
 'Driver': {'Springlake Driver'},
 'East': {'Greenlea Chase East'},
 'Expressway': {'NW 39th Expressway',
                'Northwest 39th Expressway',
                'Northwest Expressway'},
 'Extension': {'Broadway Extension', 'North Broadway Extension'},
 'F': {'W Hefner Rd STE F'},
 'Frontage': {'West I-35 Frontage'},
 'Harvey': {'North Harvey'},
 'Hiwassee': {'N Hiwassee'},
 'Hiwasww': {'N Hiwasww'},
 'Lincoln': {'N Lincoln'},
 'Ln': {'Pink Ln'},
 'MacArthur': {'N MacArthur'},
 'Main': {'Main'},
 'Maple': {'Maple'},
 'NE': {'Edmond Road NE', '108th Avenue NE', '12th Avenue NE'},
 'Northeast': {'Mustang Road Northeast', '12th Avenue Northeast'},
 'Northwest': {'24th Avenue Northwest',
               '36th Avenue Northwest',
               'Monroe Avenue Northwest'},
 'Paseo': {'Paseo'},
 'Pkwy': {'Shedeck Pkwy'},
 'Pl': {'Sonny Blues Pl', 'SOnny Blues Pl'},
 'Porter': {'North Porter'},
 'Rd': {'Bywater Rd',
        'Council Heights Rd',
        'E Interstate 240 Service Rd',
        'N Mustang Rd',
        'Ole McDonald Rd',
        'S Country Club Rd',
        'South Sooner Rd',
        'W Britton Rd',
        'W Covell Rd',
        'W Danforth Rd',
        'W I-35 Frontage Rd',
        'W Memorial Rd',
        'W. Covell Rd',
        'Wellington Rd',
        'West I 35 Frontage Rd',
        'West Memorial Rd'},
 'Rd.': {'N. Council Rd.', 'N. Mustang Rd.'},
 'Row': {'Sovereign Row'},
 'STREET': {'NW 164TH STREET', 'NW 150TH STREET'},
 'Spruce': {'Blue Spruce'},
 'St': {'2nd St',
        '3 NE 8th St',
        'Bond St',
        'E 19th St',
        'E 2nd St',
        'E Main St',
        'East 2nd St',
        'NW 157th St',
        'NW 2nd St',
        'S Broadway St',
        'SE 2nd St',
        'SE 89th St',
        'SW 12th St',
        'SW 19th St',
        'SW 47th St',
        'SW 5th St'},
 'St,': {'NW 95th St,'},
 'St.': {'E. Robinson St.', 'NW 39th St.', 'SW 24th St.'},
 'Steet': {'East Hayes Steet'},
 'Terr': {'Bell Tolls Terr'},
 'Terrace': {'Maehs Terrace',
             'North Canadian Terrace',
             'North Mitchell Terrace',
             'Northeast 20th Terrace',
             'Northeast 23rd Terrace',
             'Northwest 112th Terrace',
             'Northwest 166th Terrace',
             'Northwest 167th Terrace',
             'Northwest 220th Terrace',
             'Northwest 69 Terrace',
             'Northwest 69th Terrace',
             'Southeast 85th Terrace',
             'Southeast 86th Terrace',
             'Southeast 87th Terrace',
             'Southeast 88th Terrace',
             'Southeast 89th Terrace',
             'West Northway Terrace'},
 'Trail': {'Rock Creek Trail', 'Southeast 29th St Trail'},
 'Turnpike': {'John Kilpatrick Turnpike'},
 'Way': {'Gage Grove Way',
         'North Elk Way',
         'Ponderosa Way',
         'Rose In Bloom Way',
         'Ryan Way',
         'West Hunters Court Way',
         'West Vida Way'}}
```

There were some small inconsistencies found during that audit, but the formatting of the dataset was pretty clean.

Updating Street Names

Our next bit of code will attempt to update the street names.

In [16]:

```python
import xml.etree.cElementTree as ET
import pprint
import re
from collections import defaultdict

datadir = "data"
```

```
datafile = "Oklahoma_City.osm"

street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane
            "Trail", "Parkway", "Commons","East", "North", "West","South"]

def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)


def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")


def audit(osmfile):
    osm_file = open(osmfile, "r", errors = 'ignore')
    street_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_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
    osm_file.close()
    return street_types

Oklahoma_City_street_types = audit(datafile)

#Map the abbreviations to the expected types
MAPPING = { "St": "Street",
            "ST.": "Street",
            "STREET":"Street",
            "ST": "Street",
            "Rd.": "Road",
            "Rd": "Road",
            "RD": "Road",
            "Ave": "Avenue",
            "E":"East",
            "Ln":"Lane",
            "N":"North"
            }

def update_street(name, mapping=MAPPING):
    m = street_type_re.search(name)
    if m.group() in mapping:
        boundaries = re.compile(r'\b' + m.group() + r'$')
        name = re.sub(boundaries, mapping[m.group()], name)

    return name

for street_type, ways in Oklahoma_City_street_types.items():
    for name in ways:
        better_name = update_street(name, mapping=MAPPING)
        print(name, "=>", better_name)
```

```
NW 95th St, => NW 95th St,
Maple => Maple
North May Ave => North May Avenue
South Portland Ave => South Portland Avenue
7700 North Hudson Ave => 7700 North Hudson Avenue
S Western Ave => S Western Avenue
S Pennsylvania Ave => S Pennsylvania Avenue
S Yukon Ave => S Yukon Avenue
N Western Ave => N Western Avenue
S. Meridian Ave => S. Meridian Avenue
W. Sheridan Ave => W. Sheridan Avenue
S Rock Island Ave => S Rock Island Avenue
N Meridian Ave => N Meridian Avenue
South Robinson Ave => South Robinson Avenue
Garth Brooks Blvd => Garth Brooks Blvd
S Douglas Blvd => S Douglas Blvd
N MacArthur Blvd => N MacArthur Blvd
N Midwest Blvd => N Midwest Blvd
William Penn Blvd => William Penn Blvd
Highland Park Blvd => Highland Park Blvd
Sonny Blues Pl => Sonny Blues Pl
SOnny Blues Pl => SOnny Blues Pl
Bell Tolls Terr => Bell Tolls Terr
West Vida Way => West Vida Way
North Elk Way => North Elk Way
Gage Grove Way => Gage Grove Way
Rose In Bloom Way => Rose In Bloom Way
Ponderosa Way => Ponderosa Way
West Hunters Court Way => West Hunters Court Way
Ryan Way => Ryan Way
South Walker Ave. => South Walker Ave.
W. Sheridan Ave. => W. Sheridan Ave.
East California Ave. => East California Ave.
Paseo => Paseo
6604 Willowridge Dr. => 6604 Willowridge Dr.
Avondale Dr. => Avondale Dr.
West Lake Hefner Dr. => West Lake Hefner Dr.
NORTH PENNSYLVANIA AVE => NORTH PENNSYLVANIA AVE
NORTH MAY AVE => NORTH MAY AVE
NW 164TH STREET => NW 164TH Street
NW 150TH STREET => NW 150TH Street
Edmond Road NE => Edmond Road NE
108th Avenue NE => 108th Avenue NE
12th Avenue NE => 12th Avenue NE
36th Avenue Northwest => 36th Avenue Northwest
Monroe Avenue Northwest => Monroe Avenue Northwest
24th Avenue Northwest => 24th Avenue Northwest
Northwest 39th Expressway => Northwest 39th Expressway
NW 39th Expressway => NW 39th Expressway
Northwest Expressway => Northwest Expressway
Riverwalk DR => Riverwalk DR
Northwest 165th Circle => Northwest 165th Circle
Halley Circle => Halley Circle
Foxwood Circle => Foxwood Circle
Bradford Circle => Bradford Circle
Tioga Circle => Tioga Circle
Replublic Circle => Replublic Circle
Maehs Circle => Maehs Circle
North Geary Circle => North Geary Circle
Pinyon Circle => Pinyon Circle
Scirocco Circle => Scirocco Circle
Republic Avenue Circle => Republic Avenue Circle
North Classen Circle => North Classen Circle
South Broadway => South Broadway
N. Broadway => N. Broadway
S Broadway St => S Broadway Street
```

```
E 2nd St => E 2nd Street
East 2nd St => East 2nd Street
E Main St => E Main Street
3 NE 8th St => 3 NE 8th Street
SW 5th St => SW 5th Street
NW 2nd St => NW 2nd Street
SW 19th St => SW 19th Street
SW 47th St => SW 47th Street
SE 2nd St => SE 2nd Street
SE 89th St => SE 89th Street
Bond St => Bond Street
E 19th St => E 19th Street
NW 157th St => NW 157th Street
2nd St => 2nd Street
SW 12th St => SW 12th Street
Sovereign Row => Sovereign Row
Main => Main
Lanes Crossing => Lanes Crossing
North Robinson Avenue Suite 2000 => North Robinson Avenue Suite 2000
Tinker Diagonal => Tinker Diagonal
North Porter => North Porter
E Interstate 240 Service Rd => E Interstate 240 Service Road
W Covell Rd => W Covell Road
Ole McDonald Rd => Ole McDonald Road
W Britton Rd => W Britton Road
N Mustang Rd => N Mustang Road
S Country Club Rd => S Country Club Road
South Sooner Rd => South Sooner Road
West I 35 Frontage Rd => West I 35 Frontage Road
West Memorial Rd => West Memorial Road
W Memorial Rd => W Memorial Road
Wellington Rd => Wellington Road
W. Covell Rd => W. Covell Road
W Danforth Rd => W Danforth Road
W I-35 Frontage Rd => W I-35 Frontage Road
Council Heights Rd => Council Heights Road
Bywater Rd => Bywater Road
Mustang Road Northeast => Mustang Road Northeast
12th Avenue Northeast => 12th Avenue Northeast
Outlet Shops Dr => Outlet Shops Dr
Babcock Dr => Babcock Dr
Cornwell Dr => Cornwell Dr
Northwest Expressway Ste 1017 => Northwest Expressway Ste 1017
Northwest 69 Terrace => Northwest 69 Terrace
Northwest 220th Terrace => Northwest 220th Terrace
Northwest 167th Terrace => Northwest 167th Terrace
Northwest 69th Terrace => Northwest 69th Terrace
Southeast 89th Terrace => Southeast 89th Terrace
West Northway Terrace => West Northway Terrace
Northeast 20th Terrace => Northeast 20th Terrace
North Mitchell Terrace => North Mitchell Terrace
Northwest 112th Terrace => Northwest 112th Terrace
Southeast 88th Terrace => Southeast 88th Terrace
Maehs Terrace => Maehs Terrace
North Canadian Terrace => North Canadian Terrace
Southeast 86th Terrace => Southeast 86th Terrace
Southeast 85th Terrace => Southeast 85th Terrace
Southeast 87th Terrace => Southeast 87th Terrace
Northeast 23rd Terrace => Northeast 23rd Terrace
Northwest 166th Terrace => Northwest 166th Terrace
W Hefner Rd STE F => W Hefner Rd STE F
Highway 66 => Highway 66
East Highway 66 => East Highway 66
N Classen Blvd #103 => N Classen Blvd #103
E. Robinson St. => E. Robinson St.
```

```
NW 39th St. => NW 39th St.
SW 24th St. => SW 24th St.
NW 23rd => NW 23rd
NEW 23rd => NEW 23rd
East State Highway 152 => East State Highway 152
Pink Ln => Pink Lane
NW 150th => NW 150th
N Hiwasww => N Hiwasww
North Harvey => North Harvey
Springlake Driver => Springlake Driver
Terminal Drive, Unit 937 => Terminal Drive, Unit 937
Broadway Extension => Broadway Extension
North Broadway Extension => North Broadway Extension
South 3357 => South 3357
West I-35 Frontage => West I-35 Frontage
73069 => 73069
East Hayes Steet => East Hayes Steet
John Kilpatrick Turnpike => John Kilpatrick Turnpike
West Danforth => West Danforth
Blue Spruce => Blue Spruce
Shedeck Pkwy => Shedeck Pkwy
N MacArthur => N MacArthur
State Highway 37 => State Highway 37
South Highway 76 => South Highway 76
N. Council Rd. => N. Council Road
N. Mustang Rd. => N. Mustang Road
N Lincoln => N Lincoln
N Hiwassee => N Hiwassee
```

Now that the house cleaning items are compelte, we can move to the next phase of the analysis, CSV transfer and SQL setup.

CSV Transfer and SQL Prep

The data is now in a condition where we can import it into SQL. The XML data will be parsed and converted into tabular format and then placed into CSV files. This will enable use to import the CSV files into sqlite. (sqlprep.jpynb)

Our next script will carry out audited changes when converting to CSV.

In [17]:

```
NODES_PATH = "nodes.csv"
NODE_TAGS_PATH = "nodes_tags.csv"
WAYS_PATH = "ways.csv"
WAY_NODES_PATH = "ways_nodes.csv"
WAY_TAGS_PATH = "ways_tags.csv"

# Create Schema using the schema provided in the project instruction
SCHEMA = schema = {
    'node': {
        'type': 'dict',
        'schema': {
            'id': {'required': True, 'type': 'integer', 'coerce': int},
            'lat': {'required': True, 'type': 'float', 'coerce': float},
            'lon': {'required': True, 'type': 'float', 'coerce': float},
            'user': {'required': True, 'type': 'string'},
            'uid': {'required': True, 'type': 'integer', 'coerce': int},
            'version': {'required': True, 'type': 'string'},
            'changeset': {'required': True, 'type': 'integer', 'coerce': int},
            'timestamp': {'required': True, 'type': 'string'}
        }
    },
    'node_tags': {
        'type': 'list',
        'schema': {
            'type': 'dict',
            'schema': {
                'id': {'required': True, 'type': 'integer', 'coerce': int},
                'key': {'required': True, 'type': 'string'},
                'value': {'required': True, 'type': 'string'},
                'type': {'required': True, 'type': 'string'}
            }
        }
    },
    'way': {
        'type': 'dict',
        'schema': {
            'id': {'required': True, 'type': 'integer', 'coerce': int},
            'user': {'required': True, 'type': 'string'},
            'uid': {'required': True, 'type': 'integer', 'coerce': int},
            'version': {'required': True, 'type': 'string'},
            'changeset': {'required': True, 'type': 'integer', 'coerce': int},
            'timestamp': {'required': True, 'type': 'string'}
        }
    },
    'way_nodes': {
        'type': 'list',
        'schema': {
            'type': 'dict',
            'schema': {
                'id': {'required': True, 'type': 'integer', 'coerce': int},
                'node_id': {'required': True, 'type': 'integer', 'coerce': int},
                'position': {'required': True, 'type': 'integer', 'coerce': int}
            }
        }
    },
    'way_tags': {
        'type': 'list',
        'schema': {
            'type': 'dict',
            'schema': {
                'id': {'required': True, 'type': 'integer', 'coerce': int},
                'key': {'required': True, 'type': 'string'},
                'value': {'required': True, 'type': 'string'},
                'type': {'required': True, 'type': 'string'}
            }
        }
    }
}

# Make sure the fields order in the csvs matches the column order in the sql table sche
NODE_FIELDS = ['id', 'lat', 'lon', 'user', 'uid', 'version', 'changeset', 'timestamp']
NODE_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_FIELDS = ['id', 'user', 'uid', 'version', 'changeset', 'timestamp']
WAY_TAGS_FIELDS = ['id', 'key', 'value', 'type']
WAY_NODES_FIELDS = ['id', 'node_id', 'position']


#               Helper Functions
def get_element(osm_file, tags=('node', 'way', 'relation')):
    """Yield element if it is the right type of tag"""
    context = ET.iterparse(osm_file, events=('start', 'end'))
    _, root = next(context)
    for event, elem in context:
        if event == 'end' and elem.tag in tags:
            yield elem
            root.clear()


class UnicodeDictWriter(csv.DictWriter, object):
    """Extend csv.DictWriter to handle Unicode input"""

    def writerow(self, row):
        super(UnicodeDictWriter, self).writerow({
            k : v for k, v in row.items()
        })

    def writerows(self, rows):
        for row in rows:
            self.writerow(row)


def update_street(name, mapping):
    street=street_type_re.search(name).group()

    name=name.replace(street, mapping[street])

    return name


#clean_element function take tag['value'] and tag['key'] as input and return the update
def clean_element(tag_value, tag_key):



    ## clean street names
    if tag_key=='street':
        street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)
        full_addr=tag_value
        m = street_type_re.search(full_addr)
        if m:
            street_type = m.group() #group(): Return the string matched by the RE
            if street_type not in expected:
                if street_type in mapping:
                    tag_value=update_street(full_addr, mapping) # call update_street fu
    ## return updated tag_value
    return tag_value

## Clean and shape node or way XML element to Python dict

def shape_element(element, node_attr_fields=NODE_FIELDS, way_attr_fields=WAY_FIELDS,
                  problem_chars=problemchars, default_tag_type='regular'):

    node_attribs = {}
    way_attribs = {}
    way_nodes = []
    tags = []  # Handle secondary tags the same way for both node and way elements

    ## clean node element
    if element.tag=='node':
        for primary in element.iter():
            for i in node_attr_fields:
                if i in primary.attrib:
                    node_attribs[i]=primary.attrib[i]
        if len(element)!=0:
            for j in range(0, len(element)):
                childelem=element[j]
                tag={}
                if not problem_chars.search(childelem.attrib['k']): ## ignor problemati
                    tag["id"]=element.attrib["id"]
                    tag["type"]=default_tag_type
                    tag['value']=childelem.attrib['v']
                    if ":" in childelem.attrib['k']:
                        k_and_v=childelem.attrib['k'].split(':',1)
                        tag["type"]=k_and_v[0]
                        tag["key"]=k_and_v[1]
                        if tag["type"]=='addr':
                            tag["value"]=clean_element(tag["value"],tag["key"]) ## call
                    else:
                        tag["key"]=childelem.attrib['k']
                        if tag["type"]=='addr':
                            print(tag_value, tag["key"])
                            tag["value"]=clean_element(tag["value"],tag["key"])
                tags.append(tag)

        return ({'node': node_attribs, 'node_tags': tags})

    ## handle way element
    elif element.tag=='way':
        for primary in element.iter():
            for i in way_attr_fields:
                if i in primary.attrib:
                    way_attribs[i]=primary.attrib[i]

        if len(element)!=0:
            for j in range(0, len(element)):
                childelem=element[j]
                tag={}
                if childelem.tag=='tag':
                    if not problem_chars.search(childelem.attrib['k']):
                        tag["id"]=element.attrib["id"]
                        tag["type"]=default_tag_type
                        tag["value"]=childelem.attrib['v']
                        if ":" in childelem.attrib['k']:
                            k_and_v=childelem.attrib['k'].split(':',1)
                            tag["key"]=k_and_v[1]
                            tag["type"]=k_and_v[0]
                            if tag["type"]=='addr':
                                tag["value"]=clean_element(tag["value"],tag["key"]) #ca
                        else:
                            tag["key"]=childelem.attrib['k']
                            if tag["type"]=='addr':
                                tag["value"]=clean_element(tag["value"],tag["key"]) #up
                    tags.append(tag)

                elif childelem.tag=='nd':
                    #print (childelem.attrib['ref'])
                    way_node={}
                    way_node['id']=element.attrib['id']
                    way_node['node_id']=childelem.attrib['ref']
                    way_node['position']=j
                    #print(way_node)
                    way_nodes.append(way_node)
        return ({'way': way_attribs, 'way_nodes': way_nodes, 'way_tags': tags})

## process the file, clean and write XML into csv according to given schema

def process_map(file_in):
    """Iteratively process each XML element and write to csv(s)"""
    with codecs.open(NODES_PATH, 'w', encoding='utf-8') as nodes_file, \
         codecs.open(NODE_TAGS_PATH, 'w', encoding='utf-8') as nodes_tags_file, \
         codecs.open(WAYS_PATH, 'w', encoding='utf-8') as ways_file, \
        codecs.open(WAY_NODES_PATH, 'w', encoding='utf-8') as way_nodes_file, \
         codecs.open(WAY_TAGS_PATH, 'w', encoding='utf-8') as way_tags_file:

        nodes_writer = UnicodeDictWriter(nodes_file, NODE_FIELDS)
        node_tags_writer = UnicodeDictWriter(nodes_tags_file, NODE_TAGS_FIELDS)
        ways_writer = UnicodeDictWriter(ways_file, WAY_FIELDS)
        way_nodes_writer = UnicodeDictWriter(way_nodes_file, WAY_NODES_FIELDS)
        way_tags_writer = UnicodeDictWriter(way_tags_file, WAY_TAGS_FIELDS)


        nodes_writer.writeheader()

        node_tags_writer.writeheader()
        ways_writer.writeheader()
        way_nodes_writer.writeheader()
        way_tags_writer.writeheader()


        for element in get_element(file_in, tags=('node', 'way')):
            el = shape_element(element)
            if el:

                if element.tag == 'node':
                    nodes_writer.writerow(el['node'])
                    node_tags_writer.writerows(el['node_tags'])
                elif element.tag == 'way':
                    ways_writer.writerow(el['way'])
                    way_nodes_writer.writerows(el['way_nodes'])
                    way_tags_writer.writerows(el['way_tags'])

process_map("Oklahoma_City.osm")
```

Create the SQL Database

Now we will create our SQL database. (sqlCreate.jpynb). The the last two scripts were we some of the most frustrating to write for me. I have written larger queries in sql, but these gave me trouble.

In [18]:

```
# ref https://stackoverflow.com/questions/50735349/import-csv-into-sqlite3-insert-faile
conn=sqlite3.connect('Oklahoma_City.db')
cur = conn.cursor()
cur.execute("CREATE TABLE nodes ( id INTEGER PRIMARY KEY NOT NULL, lat REAL, lon REAL,\
    user TEXT, uid INTEGER, version INTEGER, changeset INTEGER, timestamp TEXT )")
conn.commit()
node_df = pd.read_csv('nodes.csv', dtype=object)
node_df.to_sql('nodes', conn, if_exists='append', index=False)


cur.execute("CREATE TABLE nodes_tags (\
    id INTEGER,\
    key TEXT,\
    value TEXT,\
    type TEXT,\
    FOREIGN KEY (id) REFERENCES nodes(id)\
)")
conn.commit()
nodetag_df=pd.read_csv('nodes_tags.csv')
nodetag_df.to_sql('nodes_tags', conn, if_exists='append', index=False)

cur.execute("CREATE TABLE ways (\
    id INTEGER PRIMARY KEY NOT NULL,\
    user TEXT,\
    uid INTEGER,\
    version TEXT,\
    changeset INTEGER,\
    timestamp TEXT\
)")
conn.commit()
way_df=pd.read_csv('ways.csv')
way_df.to_sql('ways', conn, if_exists='append', index=False)

cur.execute("CREATE TABLE ways_nodes (\
    id INTEGER NOT NULL,\
    node_id INTEGER NOT NULL, \
    position INTEGER NOT NULL, \
    FOREIGN KEY (id) REFERENCES ways(id),\
    FOREIGN KEY (node_id) REFERENCES nodes(id)\
)")
conn.commit()
waynode_df=pd.read_csv('ways_nodes.csv')
waynode_df.to_sql('ways_nodes', conn, if_exists='append', index=False)


cur.execute("CREATE TABLE ways_tags (\
    id INTEGER NOT NULL,\
    key TEXT NOT NULL,\
    value TEXT NOT NULL,\
    type TEXT,\
    FOREIGN KEY (id) REFERENCES ways(id)\
)")
conn.commit()
waytag_df=pd.read_csv('ways_tags.csv')
waytag_df=waytag_df.dropna(subset=['id', 'key', 'value'], how='any')
waytag_df.to_sql('ways_tags', conn, if_exists='append', index=False)
```

```
---------------------------------------------------------------------------
OperationalError                          Traceback (most recent call last)
<ipython-input-18-a40b3363715e> in <module>
      2 conn=sqlite3.connect('Oklahoma_City.db')
      3 cur = conn.cursor()
----> 4 cur.execute("CREATE TABLE nodes ( id INTEGER PRIMARY KEY NOT NULL, lat REAL, lon 
REAL,\
      5     user TEXT, uid INTEGER, version INTEGER, changeset INTEGER, timestamp TEXT
 )")
      6 conn.commit()

OperationalError: table nodes already exists
```

Exploratory Analysis

We are going to leverage sql queries that are based on the python DB-API method calls for the exploration of the dataset. For better visibility, we will create a function to execute the sql queries and return a pandas dataframe. I felt like this would put the results of the queries into a more readable format.

In [19]:

```
# importing all necessary libraries
import sqlite3
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# Style and size of graph
sns.set_style('darkgrid')
sns.set(rc={'figure.figsize':(8,6)})
```

In [20]:

```
def execute_query(QUERY, header):
    db = sqlite3.connect("Oklahoma_City.db")
    c = db.cursor()
    c.execute(QUERY)
    rows = c.fetchall()
    df = pd.DataFrame(rows, columns = header)
    db.close()
    return df
```

Lets test our database to ensure we can access the information. To do this, we will execute a query to return the count of distinct user.

In [21]:

```
#Count of distinct users
QUERY = "SELECT COUNT(DISTINCT uid) FROM nodes;"

execute_query(QUERY, ['count']).set_index([['nodes']])
```

Out[21]:

	count
nodes	1345

The query return 1,345distinct users. We now know that our sql prep and database setup worked. Now we can move forward with the analysis.

Next we will execute a query to count the nnumber of nodes, followed by a query to count the number of ways.

Count of nodes

In [22]:

```
#Count of nodes
QUERY = "SELECT COUNT(*) FROM nodes;"

execute_query(QUERY, ['count']).set_index([['nodes']])
```

Out[22]:

	count
nodes	2476566

Number of ways

In [23]:

```
#Count of ways
QUERY = "SELECT COUNT(*) FROM ways;"

execute_query(QUERY, ['count']).set_index([['ways']])
```

Out[23]:

	count
ways	263981

Our dataset contains 2476566 and 263981 ways.

Out on the town

Having grown up near Oklahoma City, I always felt as if the number of places to have dinner, or a drink were limited. We are going to see if the data confirms this by performing a count of restaurants, bars, grill and other eating and drinking establishments. We will use the LIKE command in our query. This will help us cover as much ground as possible.

In [24]:

```
#Number of places to eat or drink
QUERY = "SELECT value, COUNT(*) AS number \
FROM (SELECT key, value FROM nodes_tags UNION ALL SELECT key, value FROM ways_tags) \
WHERE key = 'amenity' \
AND value LIKE 'restaurant' \
OR value LIKE 'fast%food' \
OR value LIKE 'pub' \
OR value LIKE 'cafe' \
OR value LIKE 'bar' \
OR value LIKE 'grill' \
GROUP BY value \
ORDER BY number DESC;"

execute_query(QUERY, ['type', 'number'])
```

Out[24]:

	type	number
0	restaurant	447
1	fast_food	363
2	cafe	57
3	bar	49
4	pub	15
5	grill	1

To be honest, I am surprised that fast food is in second place, but not surprised by the low numbers of cafes, bars, and pubs.

Cycling amenities

Oklahoma City is not bicycle friendly. There are almost no biking lanes, few sidewalks, and only a couple of parks have trails wide enough to accommodate cyclists. This is an area I wanted to see grow when I lived and worked in the area.

In [25]:

```
QUERY = "SELECT value, COUNT(*) AS number \
FROM (SELECT key, value FROM nodes_tags UNION ALL SELECT key, value FROM ways_tags) \
WHERE key = 'amenity' AND value LIKE '%bicycle%'\
GROUP BY value \
ORDER BY number DESC \
LIMIT 10;"

execute_query(QUERY, ['type', 'number'])
```

Out[25]:

	type	number
0	bicycle_parking	40
1	bicycle_repair_station	5
2	bicycle_rental	1

No shock here. You have to be a brave and daring soul to cycle on the street in Oklahoma City. The downtown area is the only location for the rentals, but the vehicle traffic in the area is pretty heavy.

Public Transportation

We will stay on the transportation theme with our next query. We will look the number of bus stops for Oklahoma City. This is another area where I know the city is lacking and needs improvement.

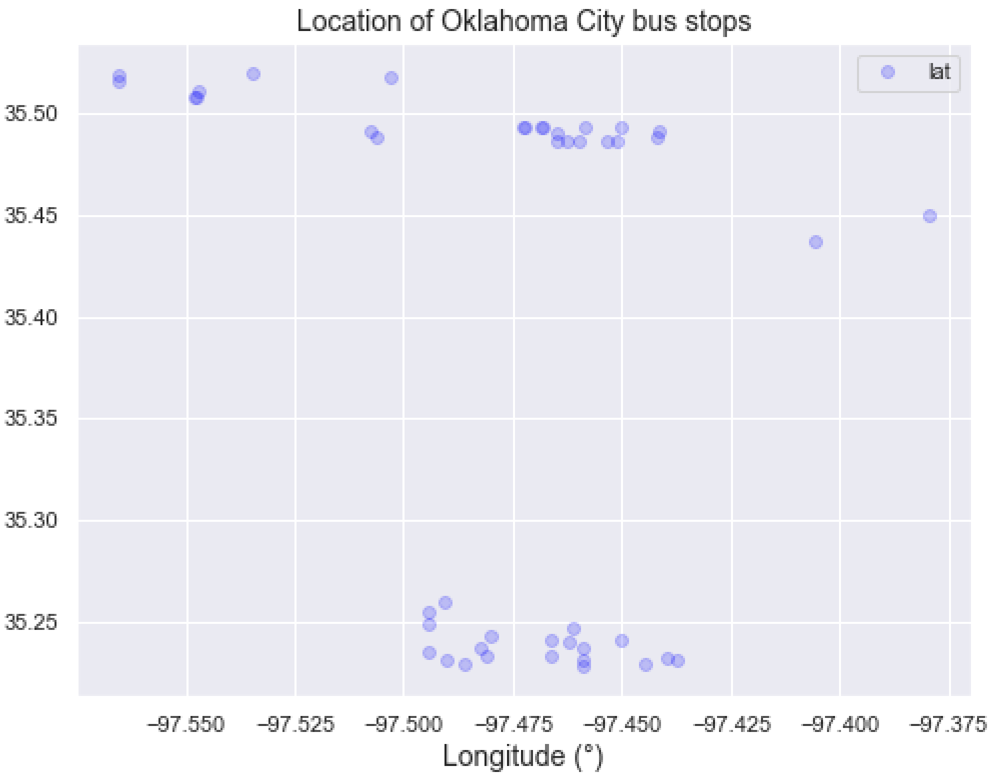
In [27]:

```
#Location of bus stops
QUERY = "SELECT nodes.lat, nodes.lon, nodes_tags.value \
FROM nodes JOIN nodes_tags ON nodes.id = nodes_tags.id \
WHERE nodes_tags.value LIKE 'bus%stop';"

df_bus = execute_query(QUERY, ['lat', 'lon', 'value'])
# Plotting the results
df_bus.plot(x = 'lon', y = 'lat', style = 'o', c = 'blue', alpha = 0.2)
plt.title('Location of Oklahoma City bus stops', fontsize = 14)
plt.xlabel('Longitude (°)', fontsize = 14)
```

Out[27]:

```
Text(0.5, 0, 'Longitude (°)')
```



Not a lot of options for people that need to get around town.

Popular Cuisine (Top 10)

Oklahoma City is a very mid western city with lots of burgers and other heavy foods. There are some ethic options in and around the city but they are greatly outnumbered by the more common american foods.

In [26]:

```
#popular cuisine
QUERY = "SELECT value, COUNT(*) AS number \
FROM (SELECT key, value FROM nodes_tags UNION ALL SELECT key, value FROM ways_tags) \
WHERE key = 'cuisine' \
GROUP BY value \
ORDER BY number DESC \
LIMIT 10;"

execute_query(QUERY, ['type', 'number'])
```

Out[26]:

	type	number
0	burger	126
1	american	56
2	sandwich	50
3	mexican	47
4	pizza	40
5	tex-mex	30
6	chinese	29
7	chicken	29

	type	number
8	coffee_shop	23
9	ice_cream	13

Cities around our target area

During our audit, we saw diffent cities in our map. In this section, we will try to locate those cities. We will need to estimate the center of those location using the mean latitude and the mean longitude. Lets give it a try.

In [28]:

```
#Other Locations
QUERY = "SELECT value, AVG(lat) as mean_lat, AVG(lon) as mean_lon, COUNT(*) as number \
FROM (SELECT id, key, value FROM nodes_tags UNION ALL SELECT id, key, value FROM ways_t
JOIN nodes ON nodes.id = all_tags.id \
WHERE key = 'city' \
GROUP BY value \
ORDER BY number DESC;"

df_cities = execute_query(QUERY,['city', 'mean_lat', 'mean_lon', 'count'])
df_cities.head()
```

Out[28]:

	city	mean_lat	mean_lon	count
0	Oklahoma City	35.483700	-97.536731	198
1	Edmond	35.650195	-97.502916	38
2	Newcastle	35.260798	-97.601581	37
3	Norman	35.240999	-97.461949	31
4	Midwest City	35.449825	-97.400207	20

Of course, Oklahoma City is our first result. Edmond is where I lived for seven years and where I went to college, the first time. My aunt and uncle live in Newcastle. Norman is the home of the University of Oklahoma. Midwest City is the location of Tinker Air Force Base.

Conclusion

- My data file from OpenStreetMap was in really good condition. I only had a few issues when working with the original file.
- This project helped me learn more about data gathering, auditing, cleaning, and analysis. This was also the first time I used sql inside of python, which is something I have been wanting to learn.
- Sadly, I did not learn much about the area where I grew up. My initial thought was to review Barnaul Russia, where my wife was born. However, that dataset had too many issues.

In [ ]: