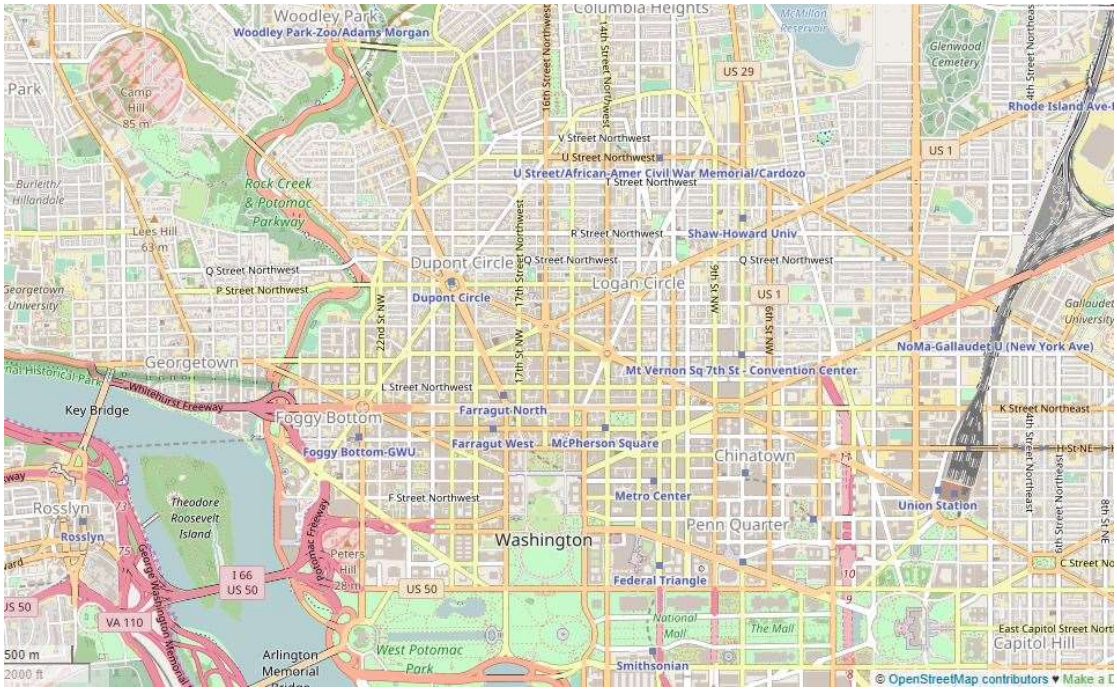
OpenStreetMap Data Case Study

By Mathias Wainer.

Map Area

Chosen map was a section of Washington D.C. Most of the area in the chosen section encompasses the Northwest quadrant of DC, this is where the national mall and most government buildings reside.



Problems Encountered in the Map

After creating a sample file of 5% of the map I ran audits for quadrant names, postal codes and cuisine types. In D.C each address specifies which quadrant it resides. It is possible to have two identical addresses in different quarters (ex: 1900 F street Northwest and 1900 F street Northeast).

Street

Snippet of street audit code:

```
osm_file = open("washington.osm", "r", encoding='utf8')

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

def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()

        street_types[street_type] += 1
```

Output of street audit:

```
Alley 2
Avenue 2
Blvd 1
BN 2
Boulevard 32
Dr 1
Drive 5
Expressway 1
floor 1
Highway 1
Hill 1
Mall 1
n.w. 1
N.W. 1
NE 23
North 1
Northeast 16541
Northwest 20854
northwest 1
NW 51
Road 1
road) 4
S.W. 1
SE 2
```

Many of the quadrants in the addresses were abbreviated, as well as a few street abbreviation errors in neighboring Virginia.

Snippet of street cleaning:

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


def audit(osmfile):
    osm_file = OSMFILE
    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

def update_name(name, mapping):
    m = street_type_re.search(name)
    if m:
        street_type = m.group()
    if street_type in mapping:
        name = re.sub(street_type_re, mapping[street_type], name)
    return name
```

Output of street cleaning changes only:

```
NE => Northeast
NW => Northwest
Dr => Drive
St => Street
SW => Southwest
SE => Southeast
```

Zip-Code

Snippet of zip-code audit:

```
post_type_re = re.compile(r'\S+\.?$', re.IGNORECASE)
postcodes = defaultdict(int)

def audit_post_type(postcodes, postcode):
    m = post_type_re.search(postcode)
    if m:
        postcode = m.group()

        postcodes[postcode] += 1

def print_sorted_dict(d):
    keys = d.keys()
    keys = sorted(keys, key=lambda s: s.lower())
    for k in keys:
        v = d[k]
        print(k, v)

def is_postcode(elem):
    return (elem.tag == "tag") and (elem.attrib['k'] == "addr:postcode")
```

Partial output of zip-code audit:

```
20001 2492
20002 10011
20003 6692
20004 24
20005 65
20005-1001 1
20005-1009 1
20005-1013 1
20005-1015 2
20005-1019 1
20005-4111 1
20005-5702 1
20005-7700 1
20006 19
20006-5346 1
```

As we can see there are no bad zip code inputs. Some have the 4-digit sub-zip code which subdivides the areas within the zip code.

Cuisine

Code showing snippet of cuisine:

```
osm_file = open("washington.osm", "r", encoding='utf8')

post_type_re = re.compile(r'\S+\.?$', re.IGNORECASE)
postcodes = defaultdict(int)

def audit_post_type(postcodes, postcode):
    m = post_type_re.search(postcode)
    if m:
        postcode = m.group()

        postcodes[postcode] += 1

def print_sorted_dict(d):
    keys = d.keys()
    keys = sorted(keys, key=lambda s: s.lower())
    for k in keys:
        v = d[k]
        print(k, v)

def is_postcode(elem):
    return (elem.tag == "tag") and (elem.attrib['k'] == "cuisine")
```

Partial output of cuisine audit:

```
Afghan 1
afghan 1
american 37
American,_Korean 1
american,pizza,beer 1
american;international;fine_dining 1
american;steak_house 1
american_and_geechy 1
asian 12
bagel 2
bagel;coffee_shop;american;breakfast;tea 1
balkan 1
Bar 1
barbecue 3
Bistro 1
Breakfast/Brunch 1
breakfast_bagels,_desserts,_pastries 1
Brunch_Restaurant 1
burger 36
```

Many equal cuisine types are divided due to capitalization. Examples are “Afghan” and “afghan”. Other errors are due to detail such as “coffee” and “coffee_shop” which should be the same.

Snippet of cuisine cleaning code:

```
def is_cuisine_name(elem):
    return (elem.attrib['k'] == "cuisine")


def audit(osmfile):
    osm_file = OSMFILE
    cuisine_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_cuisine_name(tag):
                    audit_cuisine_type(cuisine_types, tag.attrib['v'])
    osm_file.close()
    return cuisine_types

def update_name(name, mapping):
    m = cuisine_type_re.search(name)
    if m:
        cuisine_type = m.group()
    if cuisine_type in mapping:
        name = re.sub(cuisine_type_re, mapping[cuisine_type], name)
    return name
```

Output of cuisine changes only:

```
Afghan => afghan
steak => steak_house
Ethiopian => ethiopian
burger;american => burger
Korean => korean
salads => salad
Salad => salad
latin_america => latin_american
cajun;creole => cajun
coffee => coffee_shop
Diner => diner
```

Data Overview

File sizes:

washington.osm	116.03 Mb

mydb.db	71.86 MB
nodes.csv	38.55 Mb
nodes_tags.csv	3.01 Mb
ways.csv	3.9 Mb
ways_nodes.csv	1.42 Mb
ways_tags.csv	14.59 Mb

Number of nodes:

```
sqlite> SELECT COUNT(*) FROM nodes;
```

482708

Number of ways:

```
sqlite> SELECT COUNT(*) FROM ways;
```

64130

Number of unique users:

```
sqlite> SELECT COUNT(DISTINCT(e.uid)) FROM(SELECT uid FROM nodes UNION ALL
SELECT uid FROM ways) e;
```

442

Number of Chinese restaurants:

```
sqlite> SELECT nodes_tags.value, count(*)  as num
        FROM nodes_tags JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE
value='restaurant') ON nodes_tags.id=i.id
        WHERE nodes_tags.key='cuisine' and nodes_tags.value='chinese';
```

24

Looking at restaurants and fastfood:

```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
JOIN(SELECT DISTINCT(id) FROM nodes_tags WHERE value='fast_food') i
ON nodes_tags.id=i.id
WHERE nodes_tags.key ='name'
GROUP BY nodes_tags.value
```

ORDER BY num DESC

LIMIT 4;

Fast Food	Restaurant
Subway|26	Chipotle|3
McDonald's|15	Sweet Green|3
Chipotle|8	&Pizza|2
Taylor Gourmet|6	Banana Leaves|2

(restaurants would have the same query except for replacing fast_food with restaurant)

Future Problem to Fix

Some restaurants like “chipotle” are counted in both or only some braches are. This is important because DC is considered a food desert and a lot of research is being done to figure out the situation and possible solutions for a food desert. I would remove the fast_food tag completely and group everything under key: “restaurant”. Fast food is a somewhat subjective term since Taylor Gourmet is an upscale sandwich chain and there are a lot of restaurants where you pick up food at a counter that would not be considered fast foods.

The benefits of removing the fast_food key is that it becomes easier to find where Washingtonians eat. The issue with removing the fast_food tag is that we lose a little bit of detail in our map. If a researcher studying DC’s food desert wanted to look at fast food only he/she would have to query for specific restaurants such as “Subway” and “McDonalds”. With increasing levels of detail errors such as these are more likely to occur.

Conclusion

I was pleasantly surprised how clean the Washington DC dataset was. After the SQL queries ran above I learned some interesting facts about my college town. There could be some improvement in the classification of fast food and restaurants for example.