Data Wrangling Project

OpenStreetMap Data Wrangling with MongoDB

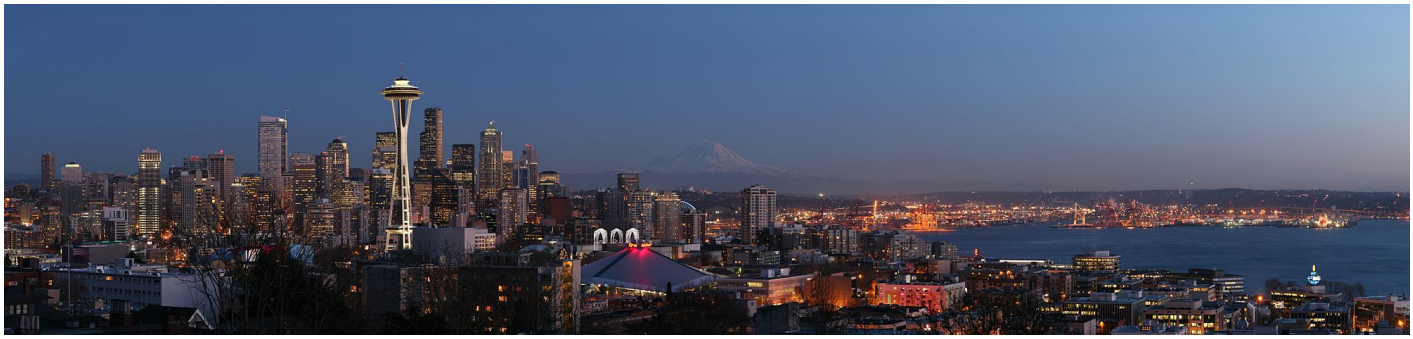
Name: Fai Alnuhait

Map area: Seattle, Washington, United States

Link: https://mapzen.com/data/metro-extracts/metro/seattle_washington/ (https://mapzen.com/data/metro-extracts/metro/seattle_washington/)

This file is structured as three sections:

- Problems encountered in the map.
- An overview of the data.
- Additional ideas about the datasets.



Seattle, or the rain city as many calls it, has been my favorite city in the US which I hope I visit one day. Its cold rainy weather is so mesmerizing, not to mention that it's the home land of several technology companies like Microsoft and Amazon. It's also the birthplace of the famous coffee chain Starbucks. Seattle is located in the west coast of the US and surrounded by water, mountains and evergreen forests. In this project, I'm aiming to fix some of the problems encountered, as well as know more about Seattle.

1. Problems encountered in the map

Solved problems

- Abbreviated directions in street names. Example: '185th Avenue SW'
- Abbreviated street names. Example: '51st Ave Northwest'
- Inconsistent postcodes. Example: '98052-4176'

Other problems

- Unclear tag names. Example: 'name_1', 'name_2'
- Data pulled from tigar database, gnis and other data sources has different layouts. Example: 'tiger:tzid', 'gnis:feature_type'

- Lacking spaces between words. Example: '20thAve.'

Direction and street names abbreviation problem

Before fixing the abbreviation problem in street names, I noticed inconsistency peoblem in directions as some are abbreviated and some are not. The following audit_directions function fix this issue. After fixing it, I edited the street names to avoid all abbreviations as can be seen below.

```
    def audit_directions(street_name):
        m = street_direc_re.search(street_name)
        if m:
            direc = m.group()
            start = m.start()
            # ensure the replacement happens only once & in the right place
            street_name = street_name[:start] + street_name[start:].replace
    (direc,mapping[direc],1)

        return street_name


    def audit_directions(street_name):
        m = street_direc_re.search(street_name)
        if m:
            direc = m.group()
            start = m.start()
            # ensure the replacement happens only once & in the right place
            street_name = street_name[:start] + street_name[start:].replace
    (direc,mapping[direc],1)

        return street_name


    def update_name(name):
        m = street_type_re.search(name)
        if m:
            street_type = m.group()
            if street_type in directions:
                new_add = name.replace(street_type,'').strip()
                m = street_type_re.search(new_add)
                if m:
                    street_type = m.group()
                    if street_type not in expected:
                        name = name.replace(street_type,mapping[street_type
    ])
            elif street_type not in expected:
                if street_type in mapping.keys():
                    name = name.replace(street_type,mapping[street_type])

        return name
```

Postcodes problem

After reviewing the different postcodes in the data, I noticed some inconsistency in the format. To solve this issue, I extracted the wanted postcode in the form of 5 digits using regular expression. The following code illustrates the process.

```
def audit_postcode(postcode):
    m = postcode_re.search(postcode)
    if m:
        postcode = m.group()

    return postcode
```

> It should be noted that some postcode followed other formats like "V8W 2L4" which were hard to standardize and fix.

2. Overview of the data

In this section, I'll list some basic statistical information about the data set.

File sizes

```
seattle_washington.osm ..... 1.76 GB
map.db ..................... 966.3 MB
nodes.csv .................. 679.4 MB
nodes_tags.csv ............. 42.7 MB
ways.csv ................... 48.1 MB
ways_tags.csv .............. 121.9 MB
ways_nodes.csv ............. 211.5 MB
```

Unique users

```
Number of users: 3623
```

Top ten contributed users

```
① Glassman:          14.88%
② SeattleImport:     8.39%
③ tylerritchie:      7.47%
④ woodpeck_fixbot:   6.48%
⑤ alester:           4.17%
⑥ Omnific:           3.98%
⑦ Glassman_Import:   2.57%
⑧ CarniLvr79:        2.41%
⑨ STBrenden:         2.32%
⑩ Brad Meteor:       2.18%
```

> As we can notice, providing data is divided among users in an acceptable way. As the top ten users contributions do not form the majority contributions but still form roughly half of it (54.85%).

The code below is used to get the number of unique users contributed to Seattle map as well as to get the top ten contributing users along with the number of information they added.

```
# Get unique users
def unique_users(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        id = element.get('uid')
        if id is not None:
            users.add(id)

    return len(users)


def top_users(filename):
    users = {}
    for _, element in ET.iterparse(filename):
        if element.get('user') is not None:
            if element.get('user') in users:
                users[element.get('user')] += 1
            else:
                users[element.get('user')] = 1

    top = list(users.values())
    total = sum(top)
    top.sort()
    top = top[-10:]

    top_ten = {}
    for k, v in users.items():
        if v in top:
            top_ten[k] = round((v/total)*100, 2)

    return sorted(top_ten.items(), key=lambda x: x[1], reverse=True)
```

Tags distribution

The following code is used to record all different tags type and their appearance.

```
Number of nodes ................... 7974362
Number of ways .................... 791249
Number of relations ............... 9892
Number of tags .................... 4474147
Number of members ................. 93658
```

> As we can see, the tags are as expected to be. Mostly consisting of nodes, tags and ways. A few relations with reasonable number of members.

```
# Recording tag names along with counting their appearance
def count_tags(filename):
    tags ={}
    for event, element in ET.iterparse(filename):
        if (element.tag in tags):
            tags[element.tag] +=1
        else:
            tags[element.tag] = 1
    return tags
```

Most popular cuisines

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT * FROM ways_tags) tags WHERE tags.key= 'cuisine' GROU
P BY tags.value ORDER BY count DESC LIMIT 10;
```

Cuisine	Count
coffee_shop	755
burger	552
mexican	478
pizza	476
sandwich	400
american	294
chinese	200
asian	189
thai	185
japanese	171

Popular Coffee shops

Since Seattle is the birthplace for the famous coffee chain Starbucks, I ran this query to examine if it would be ranked in the top ten coffee shops in Seattle.

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT *FROM ways_tags) tags
WHERE tags.id IN (SELECT DISTINCT ids.id FROM (SELECT * FROM nodes_tags
UNION SELECT * FROM ways_tags) ids WHERE (key='amenity' AND value='cafe
') OR (key='cuisine' AND value='coffee_shop') ) AND tags.key='name'
GROUP BY value ORDER BY count DESC LIMIT 10;
```

Rank	Name	Count
1	Starbucks	331
2	Tully's Coffee	19
3	BigFoot Java	15
4	Baskin-Robbins	14
4	Starbucks Coffee	14
5	Top Pot Doughnuts	13
6	Caffe Ladro	10
6	Cold Stone Creamery	10
6	Espresso	10
6	Tim Hortons	10

> Even though I expected Starbuck's to rank high in the list, the huge difference between it and other chains is a bit surprising. This list highlights the issue of inconsistency in names, as we can see Starbuck's being the first and fourth because of the naming issue. Another thing I noticed was the existence of many icecream chains in the list.

3. Additional ideas

a. Data exploration

In this section, We'll first breifly investigate the data as an exploratory step.

Amenity distribution

Here's the query to find most popular amenities in the city.

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT * FROM ways_tags) tags WHERE tags.key= 'amenity' GROU
P BY tags.value ORDER BY count DESC LIMIT 10;
```

Amenity	Count
parking	11732
restaurant	3636

bicycle_parking	3428
bench	3230
school	2456
place_of_worship	1749
fast_food	1710
cafe	1666
waste_basket	1470
fuel	1196

> Parking seems to be making a considerable proportion of the amenities in the map. Food places such as resturants, fast food ans cafes were expected.

Worship places

Investigating worship places can give us an overview of religions distribution and diversity in Seattle.

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_tags UNION SELECT *FROM ways_tags) tags WHERE tags.id IN (SELECT DISTINCT ids.id FROM (SELECT * FROM nodes_tags UNION SELECT * FROM ways_tags) ids WHERE (key='amenity' AND value='place_of_worship')) AND tags.key='religion' GROUP BY value ORDER BY count DESC;
```

Religion	Count
christian	1600
buddhist	22
jewish	20
muslim	8
unitarian_universalist	5
sikh	4
bahai	2
eckankar	2
spiritualist	2
alester	1
hindu	1
humanism	1
multifaith	1
new_thought	1

religious_science	1
scientologist	1
scientology	1
shinto	1

> This list can give us an overview of the religious diversity in Seattle as well as giving hints about the ethnicity of its locals. Religions as Scientology and Unitarian Universalist cannot really provide much information whereas Shinto for example is more of Japanese religion.

Feul stations

I ran this query to answer the question: Is there a known brand\company providing feul all over the city? or is it distributed equally among different companies?

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT *FROM ways_tags) tags WHERE tags.id IN (SELECT DISTIN
CT ids.id FROM (SELECT * FROM nodes_tags UNION SELECT * FROM ways_tags)
ids WHERE (key='amenity' AND value='fuel')) AND tags.key='name' GROUP B
Y value ORDER BY count DESC LIMIT 10;
```

Station	Count
Shell	188
Chevron	148
76	130
ARCO	76
7-Eleven	38
Texaco	32
Arco	11
Pacific Pride	10
Valero	8
local_knowledge	8

> Once again, the naming issue happened with ARCO as we can see. This data show us how four oil companies are controling the area.

Natural Places

To find out the common nature of the city, this query returns the most common natural places in Seattle.

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT * FROM ways_tags) tags WHERE tags.key= 'nature' OR ta
gs.key= 'natural' GROUP BY tags.value ORDER BY count DESC LIMIT 20;
```

Type	Count
water	6608
tree	4597
wood	2131
coastline	1987
wetland	1297
peak	908
sand	662
scrub	344
bay	322
beach	281
cliff	236
scree	163
rock	91
tree_row	88
bare_rock	82
grassland	55
saddle	54
glacier	45
spring	37
stone	33

> As expected, water occupies a large share of the natural places.

Parking fees

I was wondering how many parking whether it was bicycle or requlars parking require fees to be used compared to the free ones.

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT *FROM ways_tags) tags WHERE tags.id IN (SELECT DISTIN
CT ids.id FROM (SELECT * FROM nodes_tags UNION SELECT * FROM ways_tags)
ids WHERE (key='amenity' AND ( value='parking' OR value='bicycle_parkin
g')) ) AND tags.key='fee' GROUP BY value ORDER BY count DESC LIMIT 2;
```

- No: 3127
- Yes: 273

> This query shows that most of parking in Seattle are free. It also reveals another issue reqarding the consistency in formating. For instance: "$1.00", "$15/night", "Free", ..etc. as well as some unclear information such as "no;yes".

---

b. Additional ideas

Cities included in the map

When trying to fix the postcodes of the map, I noticed many places has postcodes located outside Seattle. To examine how information distributed among cities, the two following queries were run.

• Using city name

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT * FROM ways_tags) tags WHERE tags.key= 'city' GROUP B
Y tags.value ORDER BY count DESC LIMIT 5;
```

City	Count
Seattle	203440
Kirkland	42290
Mount Vernon	11752
Saanich	11556
Langford	2807

• Using postcode

```
sqlite> SELECT substr(tags.value,1,3) as postcode, COUNT(*) as count FR
OM (SELECT * FROM nodes_tags UNION SELECT * FROM ways_tags) tags WHERE
tags.key= 'postcode' GROUP BY postcode ORDER BY count DESC LIMIT 5;
```

City	Code	Count
Seattle (Main)	981	203966
Seattle (Vicinity)	980	47850
Bellingham	982	12615
Tacoma (Vicinity)	983	2074
Olympia	985	1224

Top sources used

The following query is used to find the top ten sources of data.

```
sqlite> SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_
tags UNION SELECT * FROM ways_tags) tags WHERE tags.key= 'source' GROUP
BY tags.value ORDER BY count DESC LIMIT 10;
```

Source	Count
King County GIS;data.seattle.gov	182895
data.seattle.gov	56301
Bing	39441
NRCan-CanVec-10.0	38132
King County GIS	34598
http://www.fs.fed.us/r6/data-library/gis/olympic/hydronet_meta.htm (http://www.fs.fed.us/r6/data-library/gis/olympic/hydronet_meta.htm)	32325
US-NPS_import_b2a6c900-5dcc-11de-8509-001e2a3ffcd7	27396
PGS	20171
tiger_import_dch_v0.6_20070830	17988
bing	12190

> The resulted list shows how formating is a big issue in the source field yet it's not easy to audit. More importantly, King County GIS, data.seattle.gov and Bing are the most common sources used in the map.

Contribution over the years

This query provide us with the total contribution made per year. To enhance its readability, I plotted the result as can be seen below.

```
sqlite> SELECT substr(tags.timestamp,1,4) as year, COUNT(*) as count FR
OM (SELECT id, timestamp FROM nodes UNION SELECT id, timestamp FROM way
s) tags GROUP BY year ORDER BY year;
```

Year	Count
2007	30713
2008	61714
2009	1424087
2010	176745
2011	176517
2012	539272

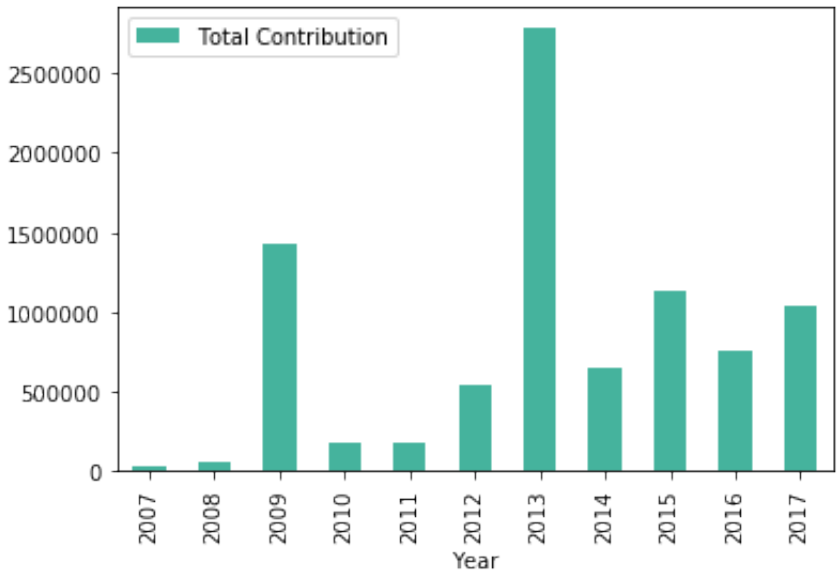
2013	2782761
2014	644481
2015	1132035
2016	761216
2017	1036070

> Since it'll be hard to understand with the numbers only, a plot representing the data is shown below.

In [20]:
```
import pandas as pd
data = pd.DataFrame(data= {'Year': ['2007', '2008', '2009', '2010',
'2011', '2012', '2013',
                                    '2014', '2015', '2016', '2017']
,
                           'Total Contribution': [30713, 61714, 142
4087, 176745, 176517,
                                                  539272, 2782761,
644481, 1132035, 761216, 1036070]})

%matplotlib inline
import matplotlib.pyplot as plt
data.plot.bar(x='Year' , y='Total Contribution' , grid=False, color
='#45B39D')
```

Out[20]: `<matplotlib.axes._subplots.AxesSubplot at 0x10349ec88>`



> The plot shows the growth of contribution over the years. The most noticeable thing is the drastic increase in 2009 and 2013 followed by returning to lower rates similar to their pervious years.

Conclusion

Standardization is the biggest issue I noticed when dealing with Seattle map. A uniform structure should be designed where every entry must be validated according to. Such structure would allow users to easily utilize the data. Users should be informed with the standard formatting they should match. Moreover, certain formats can be forced by not allowing wrong entries and using drop down lists. Although it'll be beneficial to inforce data consistency, it will be only effictive for future entries. Existed records would need to be fixed according to the designed structure specifically data pulled from other databases. Another down side of restricting entries is it might affect the number of contribution negatively.