OPEN STREET MAP PROJECT

Map area ¶

San Fransisco, CA

- https://www.openstreetmap.org/relation/396487 (https://www.openstreetmap.org/relation/396487)

I chose San Fransisco for this project, as this is my favorite city in USA and working with familiar street names and places, reminds me of my beautiful trips there. I feel fortunate to contribute in improving the data of San Fransisco on Open Street map set.

Problems in the data:

- **Overabbreviations in street names**
  Rd.,St. ctr. etc.
- **Inconsistencies in phone numbers**

  1. non- uniformity in the format of phone numbers.
  2. numbers with unicodes
  3. incorrect phone numbers-
     numbers with less than 10 digits, or too long like 20 digits
- **Inconsistencies in city names**
  There are total 446 unique entries for 'addr:city'.

  1. Numbers instead of names e.g. '11720', '155',
  2. first letter lower case e.g. san Mateo
  3. all upper case e.g. 'SAN CARLOS'
  4. non-uniform format e.g. for Oakland 7 different formats are entered. OAKLAND', 'Oakland', 'Oakland ', 'Oakland CA', 'Oakland, CA', 'Oakland, Ca', 'Okaland'
- **Inconsistencies in post codes**
  inconsistent post-codes
  CA 94607
  post codes containing letters e.g.
  M4E 2V5
  post codes not belonging to san fransisco area
  41907
  25426

In [ ]:

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

In [10]:

```
# This code creates a list of unique entries of cities in San fransisco
 map data.
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint
# functions gives the value attribute for the input element.
def get_city(element):
    return (element.attrib['v'])

# This function takes the input osm file and parse them linewise to work
 on each
#line at a time. It check's for city names in node and way tags and stor
es them in the set 'cities'.
def process_map(filename):
    cities=set()
    for _, element in ET.iterparse(filename):
        if element.tag=="node" or element.tag=="way":
            for child in element:
                # parsing for tag
                if child.tag=="tag":
                    # find key for city
                    if child.attrib["k"]=="addr:city":
                            # add the city name to set cities
                            cities.add(get_city(child))
    return cities
process_map('sfo.osm')
```

Data cleaning:

In this project, I focused on cleaned street address data and phone number data. With little modification the codes can be used for cleaning city names and postal codes.
Claning street names: In order to clean the street address data, we collect all the irregularities in the data by using audit_street_type' function. As the output we get a dictionary containing irregular, incorrect and abbrevieted names as keya and their full names in correct format as their values. Further We use this dictionary and 'update_street_name' function for rectifying the inconsistencies in street names.

In [ ]:

```
#Code to audit street names. The output contains a
#dictionary with unique entries of abbreviated names as keys and a list
 of full street names
#as values.

# this function inputs street name and
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)

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


def audit(osmfile):
    osm_file = open(osmfile, "r")
    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
```

In [6]:

```
import csv
import codecs
import pprint
import re
import xml.etree.cElementTree as ET
import cerberus
import schema
expected = ["Street", "Avenue", "Boulevard","Center", "Drive", "Court",
"Place","Plaza","Suite", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons","Way"]

# mapping variable used in update_streetname function
mapping = {
            "Ave": "Avenue",
            "Ave.": "Avenue",
            "Blvd": "Boulevard",
            "Blvd.": "Boulevard",
            "Ct" : "Court",
            "Ct.":"Court",
            "Ctr" :"Center",
            "Ctr." :"Center",
            "Dr": "Drive",
            "Dr." : "Drive",
            "Ln" : "Lane",
            "Ln." : "Lane",
            "Plz" :"Plaza",
            "Plz." :"Plaza",
            "Rd.":"Road",
            "Rd":"Road",
            "Sq" : "Square",
            "Sq." : "Square",
            "St": "Street",
            "St.": "Street",
            "st" : "Street",
            "st." : "Street",
            "Ste" : "Suite",
            "Ste." : "Suite",
            }
def update_street_name(name, mapping):
    newname=""
    text=name.split(" ")

    for i in range(len(text)):
        if text[i] in mapping:
            #print(text[i])
            text[i]=mapping[text[i]]
            break

    for i in range(len(text)):
        newname=newname+' ' + text[i]
    return newname.strip()
```

Cleaning Phone numbers: In order to clean the non-uniformity in phone numbers, all phone numbers are converted in +X (XXX) XXX-XXXX format. and the numbers with extensions are converted in to +X (XXX) XXX-XXXX ext. XXXX
The unintelligible numbers have been removed and assigned with the value "No Phone Number".

In [8]:

```
# Regex to allow the phone number in accepted format.
phone_number_re=re.compile(r'^\+1\s\([0-9]{3}\)\s[0-9]{3}\-[0-9]{4}$')

# usig regex phone_number_re following function compares the input numbe
r with regex
#and outputs boolean value True or False if the number is in accepted fo
rmat or not respectively
def audit_pn_type(number):
    # search in regex whether number is in accepted format.
    m = phone_number_re.search(number)
    if m:
        return True
    else:
        return False

# Following function checks whether input number has an extension or no
t.
def check_ext(numeric):
    if len(numeric)>10 and len(numeric)<16:
        return True
# function extracts the numeric data from the number removing any -, .,
 or letters etc.
def get_numeric(s):
    return str(filter(str.isdigit,s))
# main function inputs number in different format and outputs in accepte
d format.
def update_number(wrong_number):
    # if the number is in unicode
    if(isinstance(wrong_number,unicode)):
        #convert it to string
        wrong_number = wrong_number.encode('utf-8')
    # keep only numerics of the string wrong_number
    numeric=get_numeric(wrong_number)
    # remove country code '1' if any
    if len(numeric)==11 and numeric.startswith('1'):
        numeric=numeric[1:]
    # if number has extension put in extension format.
    if check_ext(numeric):
        newnumber='+1 ({}) {}-{} ext. {}'.format(numeric[0:3], numeric[3
:6], numeric[6:10],numeric[10:])
        return newnumber
    # if number is valid i.e. it has 10 digits convert it in acceptable
 format.
    elif len(numeric)==10:
        newnumber='+1 ({}) {}-{}'.format(numeric[0:3], numeric[3:6], num
eric[6:])
        return newnumber
    # if number does not fall in to any of above criterea remove it and
 put "no phone number" instead.
    else:
        newnumber="No Phone Number"
        return newnumber
```

Data Overview

This section provide s the size of files, some statistics based on sfo.osm file along with the SQL queries used to derive them.

Files size-

sfo.osm: 1.41 gb
nodes.csv : 557 mb
nodes_tags.csv : 9.7 mb
ways.csv : 50 mb
ways_tags.csv: 60.3 mb
ways_nodes.csv : 189.6 mb
sfo.db : 992 mb

List of tags:

{'osm': 1, 'bounds': 1, 'node': 6640188, 'tag': 2067037, 'way': 827664, 'nd': 7885162, 'relation': 7830, 'member': 63494}

Number of nodes:

```
In [ ]: SELECT count(*)
        FROM nodes;
```

Output:

6640188

Number of ways:

SELECT count(*) FROM ways;

Output:

827664

Number of unique users:

SELECT count(distinct(u.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways)u;

Output:

2892

In [ ]:

```
SELECT count(u.id) as c, u.uid
FROM
(SELECT  uid, id FROM nodes UNION ALL  SELECT uid, id FROM ways )u
GROUP BY u.uid  ORDER BY c DESC LIMIT 10 ;
```

In [ ]:

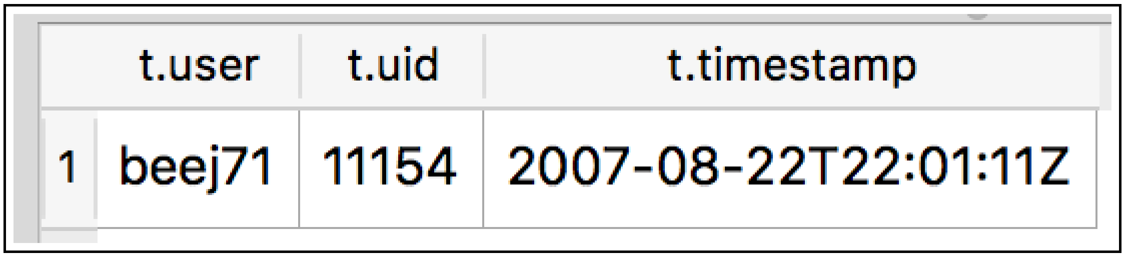
```
### Output:
<table><tr><td><img src='top10.png'></td></tr></table>
```

The first user:

In [ ]:

```
SELECT t.user, t.uid, t.timestamp
FROM (SELECT user, uid, timestamp
FROM nodes
UNION ALL
SELECT user, uid, timestamp
FROM ways) t
ORDER BY
t.timestamp
ASC LIMIT 1;
```

Output:



	t.user	t.uid	t.timestamp
1	beej71	11154	2007-08-22T22:01:11Z

Total unique entries for phones

In [ ]:

```
SELECT COUNT(DISTINCT(k.value))
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)k
WHERE k.key=='phone';
```

Output:

3220

Total unique entries for postcodes

In [ ]:

```
SELECT COUNT(DISTINCT(k.value)) <br>
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)k <br>
WHERE k.key=='postcode';
```

Output:

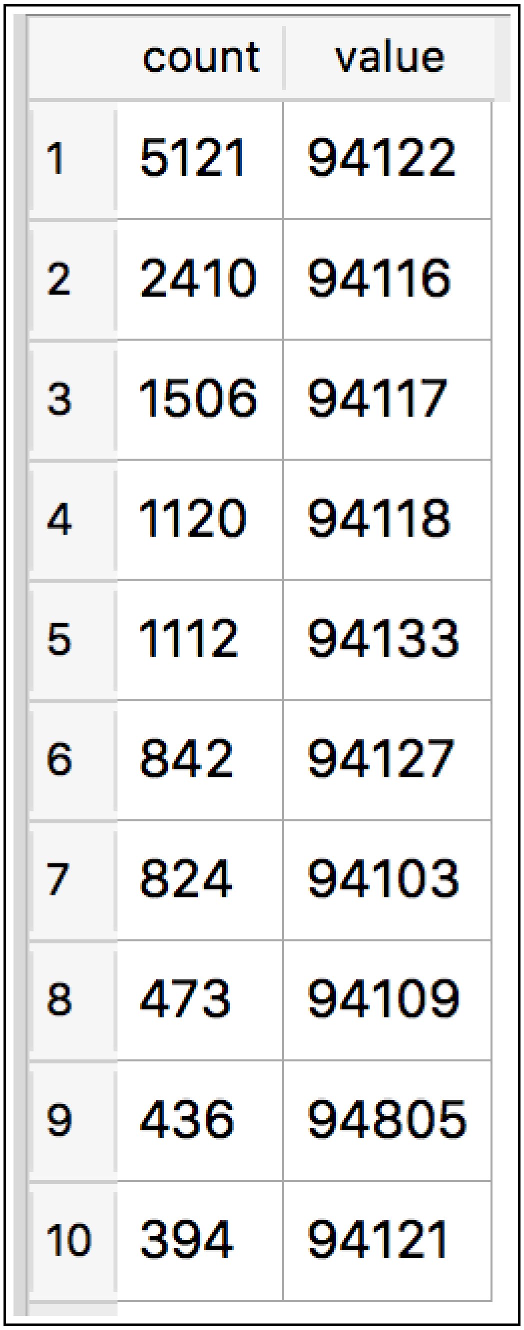
172

Top ten highest entries of post codes

In [ ]:

```
SELECT COUNT(k.id) as count, k.value  <br>
FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags)k <br>
GROUP BY k.value having k.key=='postcode' ORDER BY count DESC LIMIT 10;<br>
```

Output:



	count	value
1	5121	94122
2	2410	94116
3	1506	94117
4	1120	94118
5	1112	94133
6	842	94127
7	824	94103
8	473	94109
9	436	94805
10	394	94121

Top 10 appearing amenities:

SELECT k.value, COUNT(_) as count_
FROM
(SELECT FROM nodes_tags
UNION ALL
SELECT * FROM ways_tags)k
GROUP BY k.value
HAVING k.key='amenity'
ORDER BY count
DESC LIMIT 10;

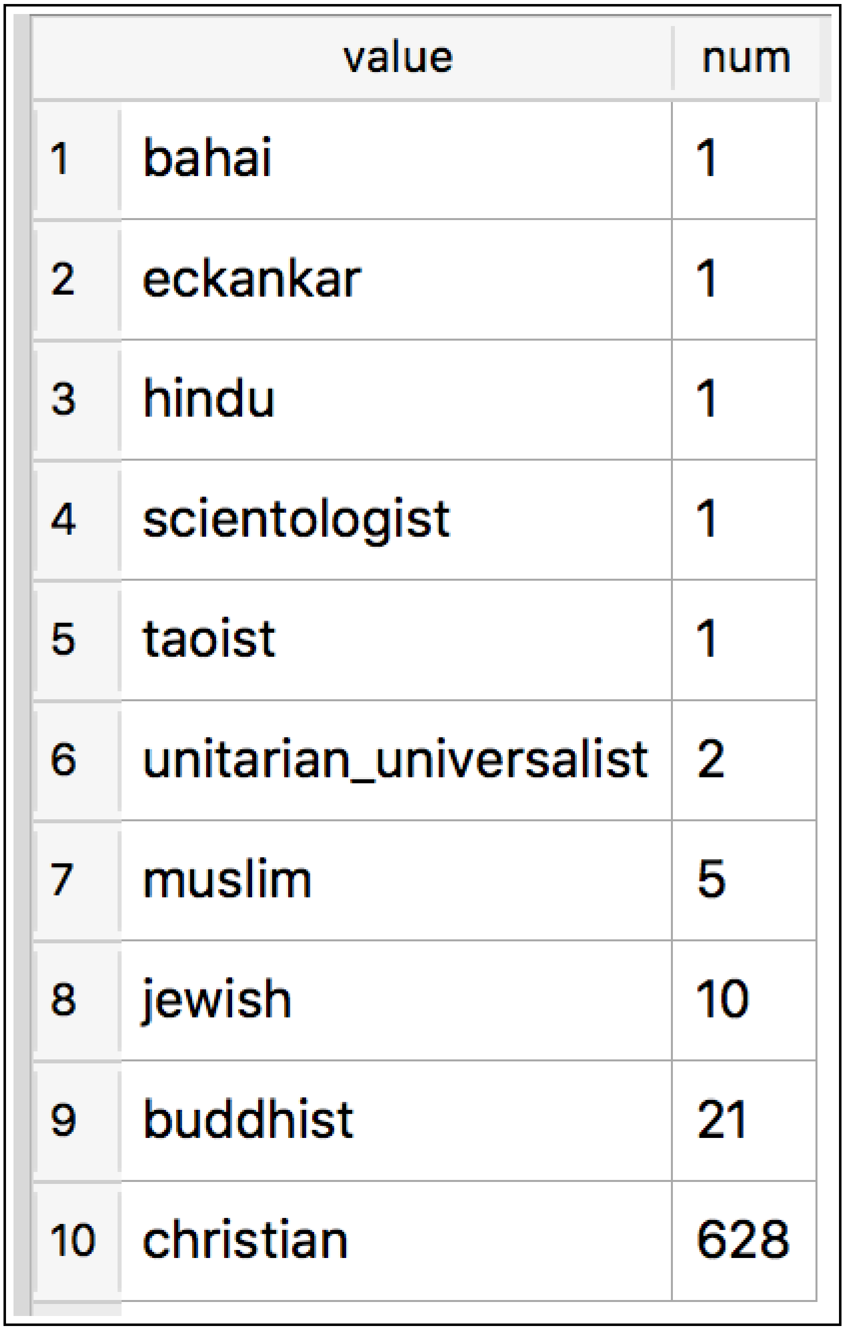
Output:

Religions:

In [ ]:

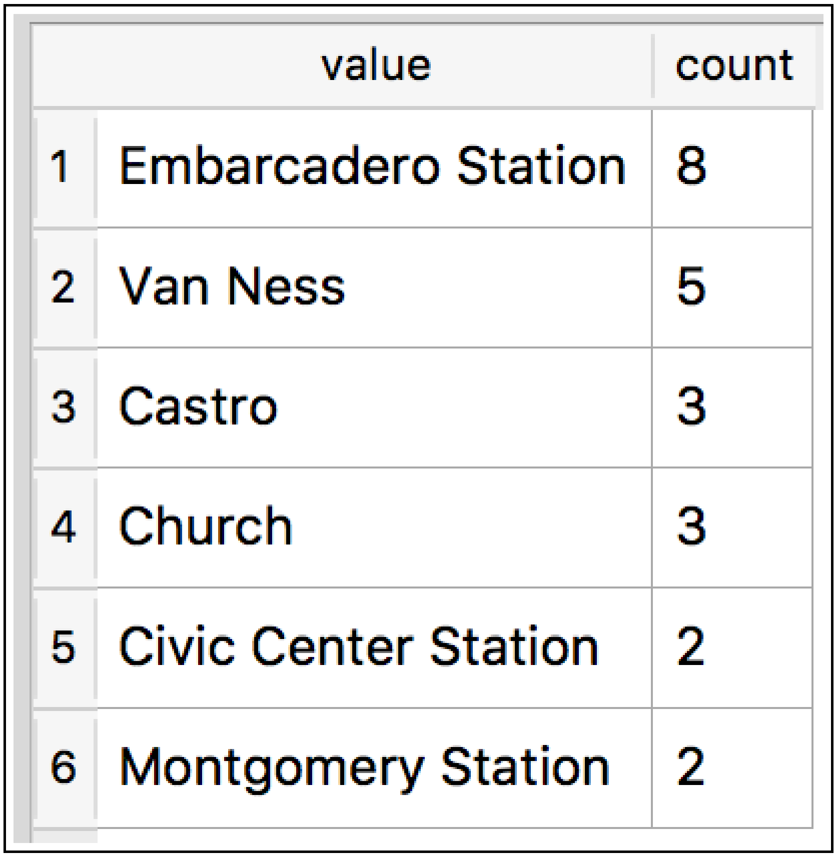
```
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='place_of_worship') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='religion'
GROUP BY nodes_tags.value
ORDER BY num ASC;
```

Output:



	value	num
1	bahai	1
2	eckankar	1
3	hindu	1
4	scientologist	1
5	taoist	1
6	unitarian_universalist	2
7	muslim	5
8	jewish	10
9	buddhist	21
10	christian	628

Network:



	value	count
1	Embarcadero Station	8
2	Van Ness	5
3	Castro	3
4	Church	3
5	Civic Center Station	2
6	Montgomery Station	2

```
In [ ]: SELECT DISTINCT(value) FROM NODES_TAGS WHERE key='network';
```

Output:

In [ ]:

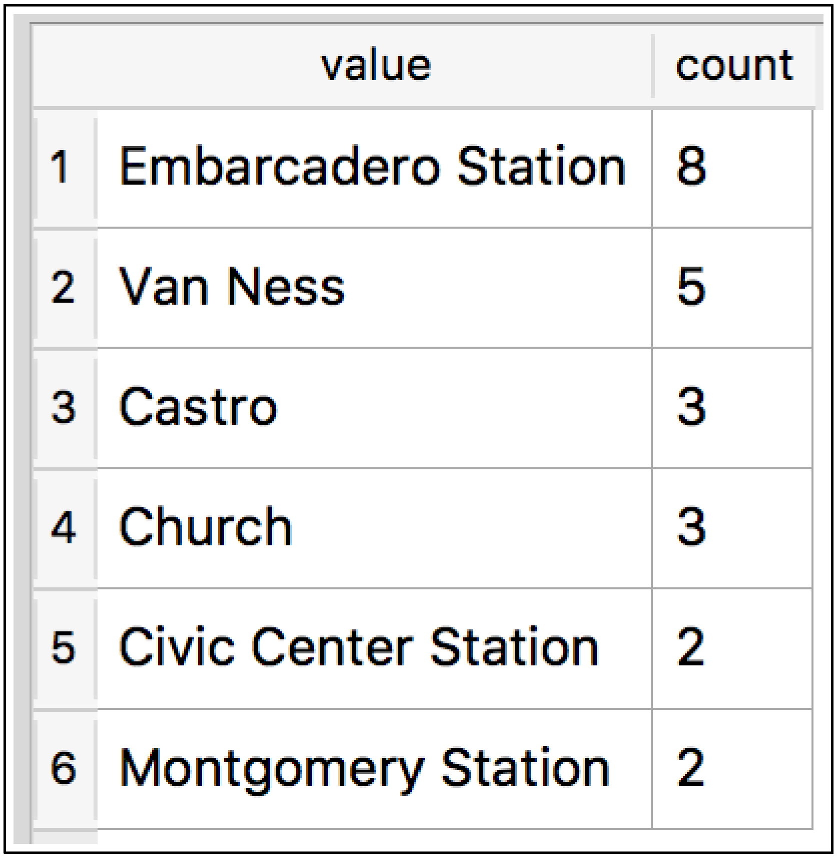
```
"BART"
"Muni"
"BART;Muni"
"1AX, 18"
"1AX"
"1"
"Caltrain"
"SFO"
"Marin Transit"
"Caltrain;BART"
"Ford GoBike"
"Caltrain;Redi-Wheels"
"Megabus"
"1, 1AX"
"1, 1AX, 18"
"SamTrans"
"zipcar"
"Golden Gate Transit"
"muni"
"Golden Gate Tansit"
"Clipper"
"american express;cirrus;co-op;discover;mastercard;plus;pulse;visa"
"AC Transit"
"Co-Op ATM"
"Union City Transit"
"CO-OP"
"MUNI"
```

Sort station names by count, descending:

In [ ]:

```
SELECT value, count(*) as count
FROM nodes_tags
GROUP BY value HAVING key='station_name'
ORDER BY count DESC;
```

Output:



	value	count
1	Embarcadero Station	8
2	Van Ness	5
3	Castro	3
4	Church	3
5	Civic Center Station	2
6	Montgomery Station	2

Ideas for additional improvements:

One significant part in the cleaning of data is avoiding duplication and loss of data and that that can be improved by standardizing the key inputs. for instance for post codes possible key inputs can be post-code, postcode, postal_code, zip-code, If the user could be restrained by inputting key values in standard way that can improve the quality by large.
Understandably, this is not an easy task, as there can be many correct notions for certain key. One way we can implement key restrainment is following. Each line that user inputs if it has attribute 'k' , it will be checked for data quality. As we have implemented a mapping dictionary for street name audit, we can create a dictionary containing possible variation of key words and their values be the acceptable key word. Let's say we use the standard term 'zip code'. If a user types a key 'postal_code' , it will automatically be input as 'zip code'.

In San Fransisco most people use public transportation for their daily commutes as well as for tourism. Therefore, the improvements in the data related to public transport would be of tremendous use. The 'network' key has following entries.

In [ ]:

```
SELECT DISTINCT(value)
FROM NODES_TAGS
WHERE KEY='network';
```

In [ ]:

```
"BART"
"Muni"
"BART;Muni"
"1AX, 18"
"1AX"
"1"
"Caltrain"
"SFO"
"Marin Transit"
"Caltrain;BART"
"Ford GoBike"
"Caltrain;Redi-Wheels"
"Megabus"
"1, 1AX"
"1, 1AX, 18"
"SamTrans"
"zipcar"
"Golden Gate Transit"
"muni"
"Golden Gate Tansit"
"Clipper"
"american express;cirrus;co-op;discover;mastercard;plus;pulse;visa"
"AC Transit"
"Co-Op ATM"
"Union City Transit"
"CO-OP"
"MUNI"
```

I found that data related to cal-trains, barts and munis needs to be added and cleaned. Again half of this problem can be sorted by standardizing key inputs. Tourist attarctions can be a feature, that many people will find useful to explore.

In [ ]:

```
SELECT nodes_tags.value
FROM nodes_tags
JOIN
(SELECT id FROM nodes_tags
WHERE value='attraction') e
ON e.id=nodes_tags.id
WHERE nodes_tags.key='name',LIMIT 10;
```

In [ ]:

```
"Inspiration Point"
"Fitzgerald Marine Reserve"
"Little Farm"
"Rotary Peace Grove"
"Alcatraz Island"
"Sather Gate"
"Golden Gate Live Steamers"
"Jepson Laurel"
"Painted Ladies"
"Pinchot Tree"
```

In [ ]:

```
For some reasons this data also include animal names as tourism attraction.
Presuming that these datas are details of some zoo attractions list,
it should not be kept with the 'k'='tourism','v'='attraction'.
```

Conclusion:

The open street map data of city of San-Fransisco is vast and detailed and encorporates various aspects of this beautiful city. Still more cleaned data can be added for the utilization of tourists. In this project, I have cleaned the street address and phone number data. Still post codes, city names, network, railway datas require thorough cleaning. The data of tourist attractions is quite clean and uniform but still they have some entries that will not be qualified as tourist attraction of San Fransisco.
I did notice, standardizing the key values can help users to clean and analyze the data easily and will save a lot of time.

In [ ]: