Open Street Map Project: San Diego

Intro

OpenStreetMap (OSM) is an open-source project attempting to create a free map of the world (think Google Maps, but not by Google).

The data is gathered and entered voluntarily. It is maintained by the OpenStreetMap foundation and is a collaborative effort with > 2 million contributors. OpenStreetMap data is freely available to download in many formats.

OSM presents an ideal opportunity to practice data wrangling since:

- The data is user-generated → there will be a significant quantity of errors/ not stringent data
- The data is free to download in many formats including XML
- The data is relatable and human-understandable because it represents real places

I decided to work with the area of San Diego because I spend a semester abroad there. Specifically, I lived in the Pacific Beach area. I thought it would be nice to explore the city through the data after many hours spent experiencing the city in the real world.

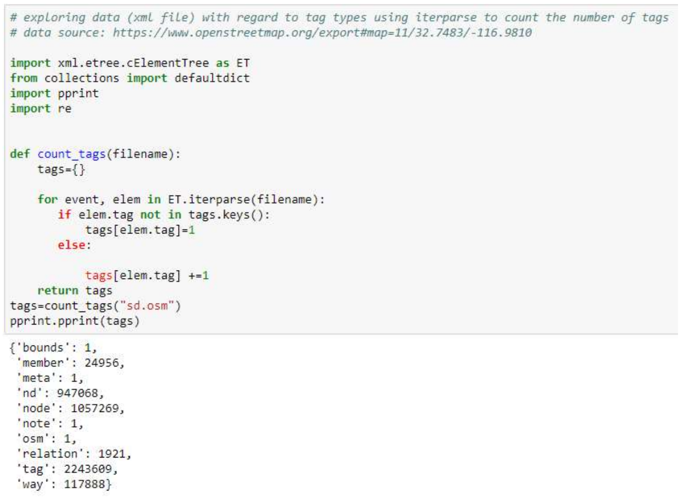
Data source: https://www.openstreetmap.org/export#map=11/32.7483/-116.9810

Data Audit

The first step was to download the map as an XML file. Looking at the XML file, we see that it uses different types of tags. I did this parsing my dataset and using ElementTree and then count the number of unique tags.

There are three core, top-level elements in OSM:

- *Nodes* represent a single point and have an id, latitude, and longitude.
- *Ways* are made up of ordered lists of nodes that describe a linear feature such as a trail, or an area such as a park. They contain a list of the nodes that comprise the way as well as tags for detailed information.
- *Relations* are comprised of an ordered list of members that can be either nodes or ways. They are used to represent logically or geographic relationships between features and contain a list of members as well as tags describing the element.

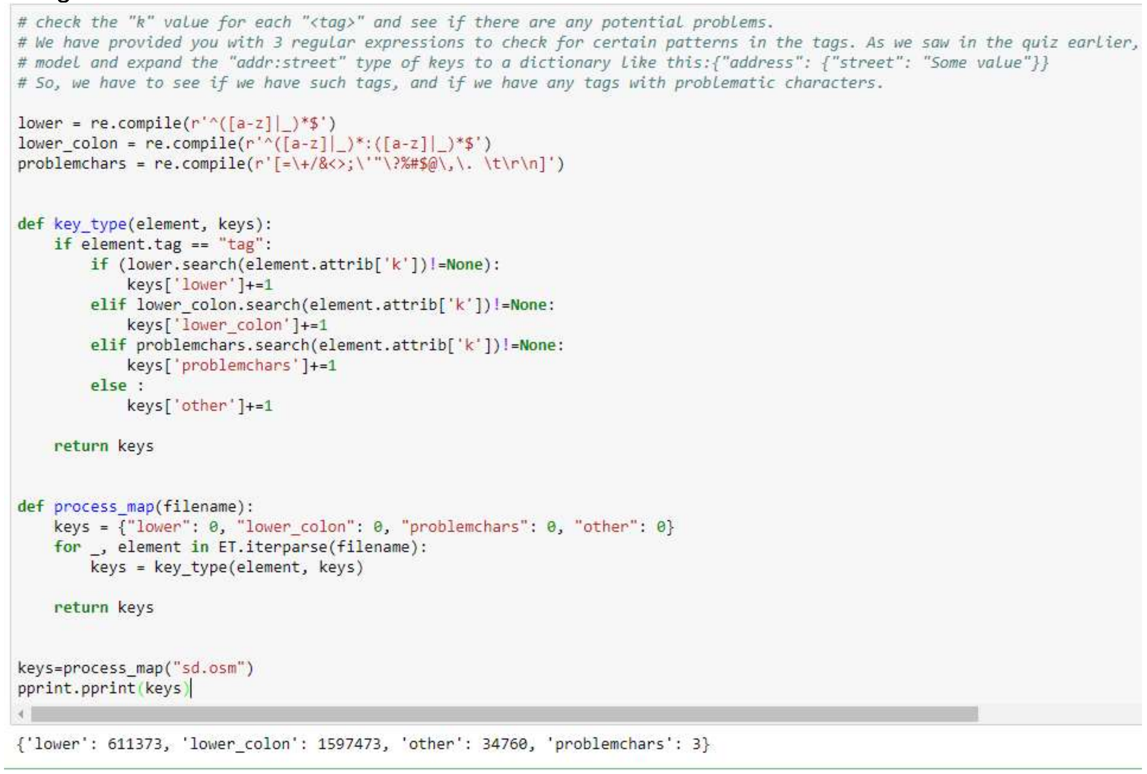


```
# exploring data (xml file) with regard to tag types using iterparse to count the number of tags
# data source: https://www.openstreetmap.org/export#map=11/32.7483/-116.9810

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


def count_tags(filename):
    tags={}

    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("sd.osm")
pprint.pprint(tags)
```

```
{'bounds': 1,
 'member': 24956,
 'meta': 1,
 'nd': 947068,
 'node': 1057269,
 'note': 1,
 'osm': 1,
 'relation': 1921,
 'tag': 2243609,
 'way': 117888}
```

Explore patterns in the tags

The "k" value of each tag contains various patterns. Using the 3 regular expressions, we can check for certain patterns in the tags. Here, I counted the frequency of each of the tag categories:



```
# check the "k" value for each "<tag>" and see if there are any potential problems.
# We have provided you with 3 regular expressions to check for certain patterns in the tags. As we saw in the quiz earlier,
# model and expand the "addr:street" type of keys to a dictionary like this:{"address": {"street": "Some value"}}
# So, we have to see if we have such tags, and if we have any tags with problematic characters.

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

    return keys


keys=process_map("sd.osm")
pprint.pprint(keys)
```

```
{'lower': 611373, 'lower_colon': 1597473, 'other': 34760, 'problemchars': 3}
```

Data Cleaning

For the data cleaning, I decided on the following approach, adapted from the Udacity course on Data Wrangling:

1. Identify the causes of any inconsistent/incorrect data

2. Develop a set of corrective cleaning actions

3. Implement the data cleaning plan: run cleaning scripts and transfer the cleaned data to .csv files

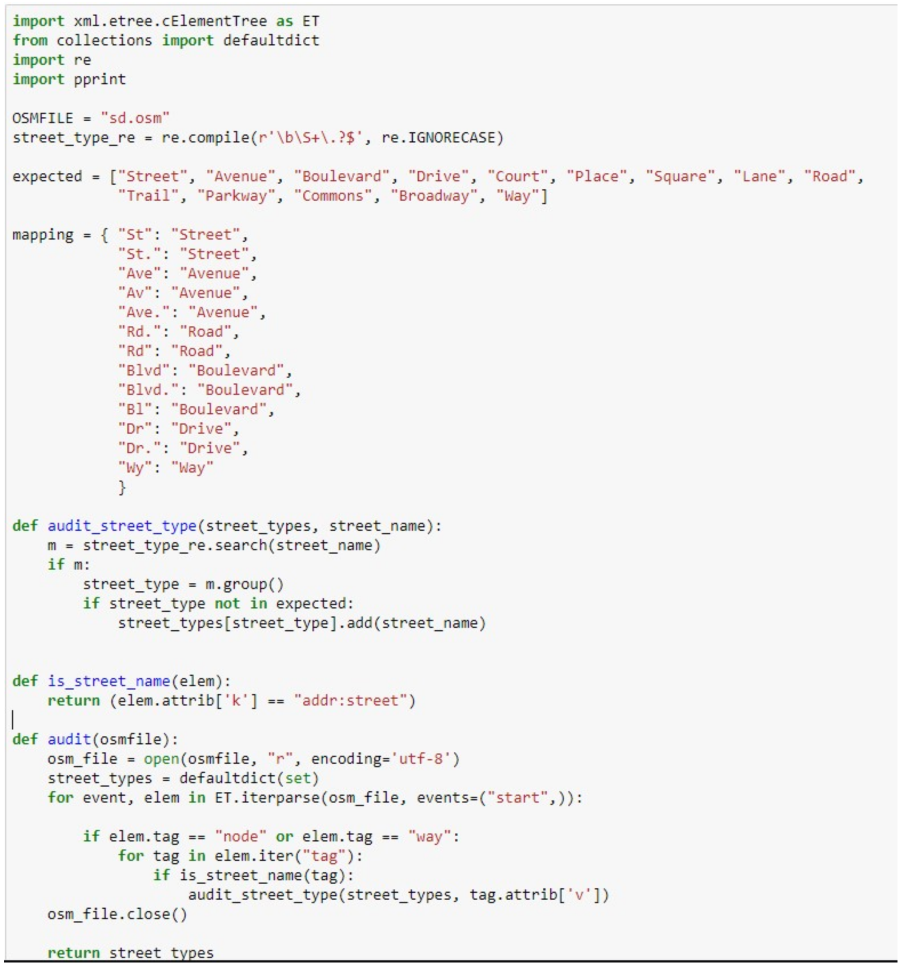
4. Import the data from .csv files to a SQL database and perform SQL queries on the data to identify any further inconsistencies that would necessitate returning to step 1.

Data wrangling is an iterative procedure, and as such, I expected that I might need to work through these steps several times.

Street Types

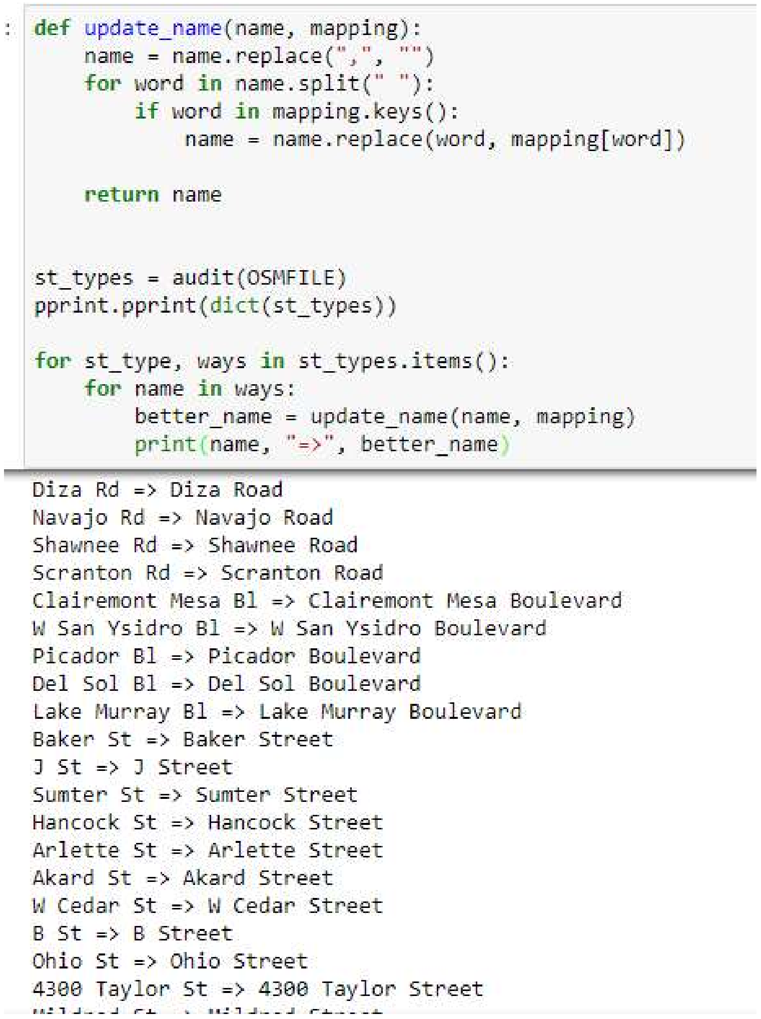
The primary variable I could audit for validity and consistency was the street names associated with the node and way tag.

The main problem here is that there are a lot of inconsistencies in the street names. From my initial exploratory examination of the data, I had noticed a wide variety in street name endings and abbreviations.



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

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

expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons", "Broadway", "Way"]

mapping = { "St": "Street",
            "St.": "Street",
            "Ave": "Avenue",
            "Av": "Avenue",
            "Ave.": "Avenue",
            "Rd.": "Road",
            "Rd": "Road",
            "Blvd": "Boulevard",
            "Blvd.": "Boulevard",
            "Bl": "Boulevard",
            "Dr": "Drive",
            "Dr.": "Drive",
            "Wy": "Way"
            }

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", encoding='utf-8')
    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 order to standardize the street names, I created a mapping and applied it to the data during the conversion from XML to CSV files.


```
def update_name(name, mapping):
    name = name.replace(",", "")
    for word in name.split(" "):
        if word in mapping.keys():
            name = name.replace(word, mapping[word])

    return name


st_types = audit(OSMFILE)
pprint.pprint(dict(st_types))

for st_type, ways in st_types.items():
    for name in ways:
        better_name = update_name(name, mapping)
        print(name, "=>", better_name)
```

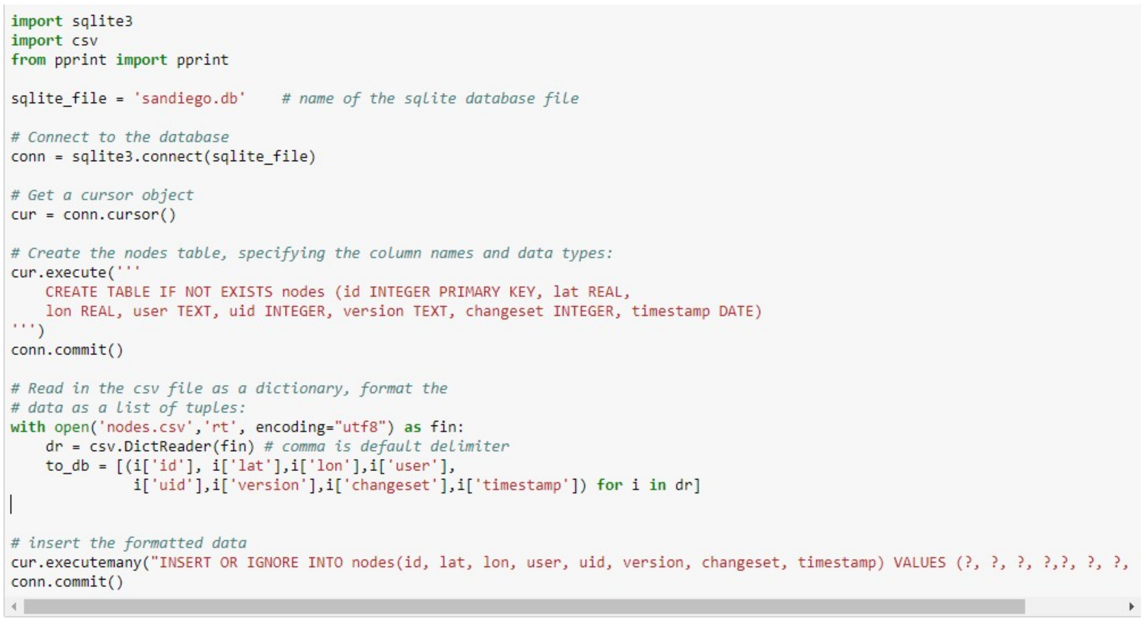
```
Diza Rd => Diza Road
Navajo Rd => Navajo Road
Shawnee Rd => Shawnee Road
Scranton Rd => Scranton Road
Clairemont Mesa Bl => Clairemont Mesa Boulevard
W San Ysidro Bl => W San Ysidro Boulevard
Picador Bl => Picador Boulevard
Del Sol Bl => Del Sol Boulevard
Lake Murray Bl => Lake Murray Boulevard
Baker St => Baker Street
J St => J Street
Sumter St => Sumter Street
Hancock St => Hancock Street
Arlette St => Arlette Street
Akard St => Akard Street
W Cedar St => W Cedar Street
B St => B Street
Ohio St => Ohio Street
4300 Taylor St => 4300 Taylor Street
```

SQL Database

When the San Diego OSM data was converted from the original source XML file to 5 different CSV files the first data cleaning precudure was conducted. In the conversion process, the data was validated against a predefined schema.py file to ensure that both the structure of the csv files and the types of the data entered were as expected (according to the defined schema).

This procedure took me some time and was finally successfully completed after several tries and some researching plus reworking of the cleaning and converting scripts. The main problem was that I am using Python 3 and the Lesson sample code is written for Python 2.

The next step was to initialize a SQL database and to create the tables within based on a pre-defined schema.



```
import sqlite3
import csv
from pprint import pprint

sqlite_file = 'sandiego.db'    # name of the sqlite database file

# Connect to the database
conn = sqlite3.connect(sqlite_file)

# Get a cursor object
cur = conn.cursor()

# Create the nodes table, specifying the column names and data types:
cur.execute('''
    CREATE TABLE IF NOT EXISTS nodes (id INTEGER PRIMARY KEY, lat REAL,
    lon REAL, user TEXT, uid INTEGER, version TEXT, changeset INTEGER, timestamp DATE)
''')
conn.commit()

# Read in the csv file as a dictionary, format the
# data as a list of tuples:
with open('nodes.csv','rt', encoding="utf8") as fin:
    dr = csv.DictReader(fin) # comma is default delimiter
    to_db = [(i['id'], i['lat'],i['lon'],i['user'],
              i['uid'],i['version'],i['changeset'],i['timestamp']) for i in dr]


# insert the formatted data
cur.executemany("INSERT OR IGNORE INTO nodes(id, lat, lon, user, uid, version, changeset, timestamp) VALUES (?, ?, ?, ?,?, ?, ?,
conn.commit()
```

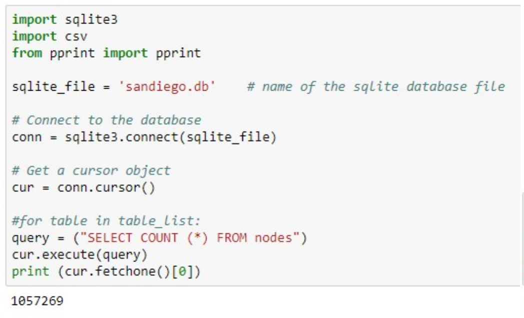
Exploring the SQL Database

I then imported the data from the CSV files into the corresponding SQL tables. I encountered a few issues along the way, mainly with the type of data and the encoding.

After successfully importing all the data into the tables, it was time to start the most rewarding phase of the project, investigating the data! I formulated a few questions to guide my exploration of the dataset:

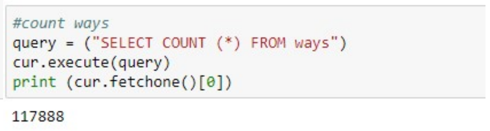
First some general statistics:

The number of nodes is 1,057,269 and the number of ways tags is 1,117,888.



```
import sqlite3
import csv
from pprint import pprint

sqlite_file = 'sandiego.db'    # name of the sqlite database file

# Connect to the database
conn = sqlite3.connect(sqlite_file)

# Get a cursor object
cur = conn.cursor()

#for table in table_list:
query = ("SELECT COUNT (*) FROM nodes")
cur.execute(query)
print (cur.fetchone()[0])
```

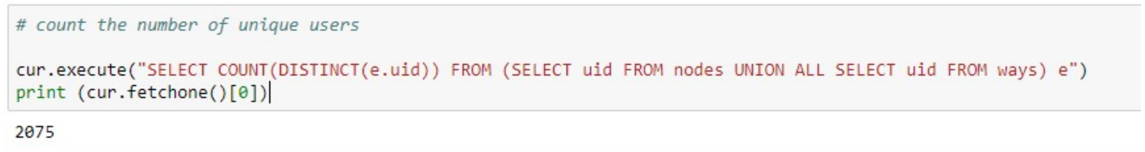
```
1057269
```



```
#count ways
query = ("SELECT COUNT (*) FROM ways")
cur.execute(query)
print (cur.fetchone()[0])
```

```
117888
```

The number of unique users is 2,075.



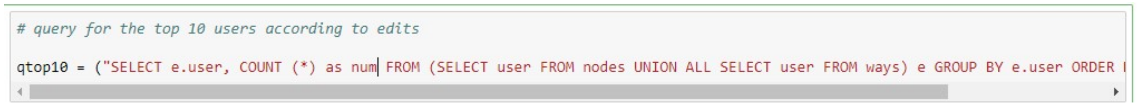
```
# count the number of unique users

cur.execute("SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e")
print (cur.fetchone()[0])
```

```
2075
```

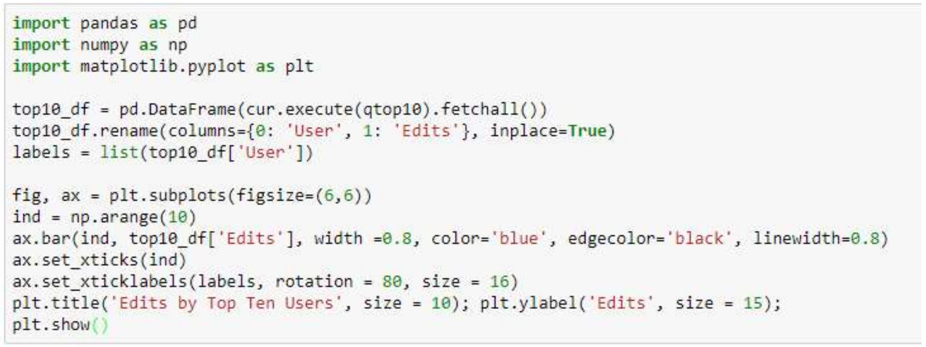
1. Who contributed the most edits to the dataset?

In order to find out the users who contributed the most to the dataset I wrote this query.

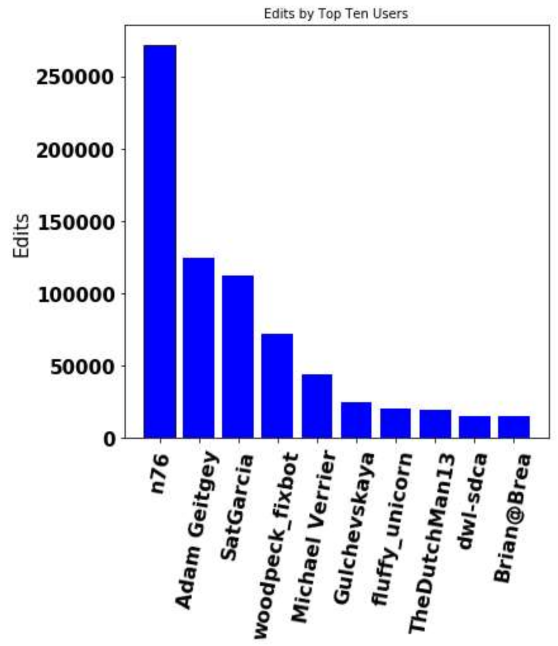
Additionally, I displayed the top 10 users in a graph for a better visualization of the data.



```
# query for the top 10 users according to edits

qtop10 = ("SELECT e.user, COUNT (*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e GROUP BY e.user ORDER
```

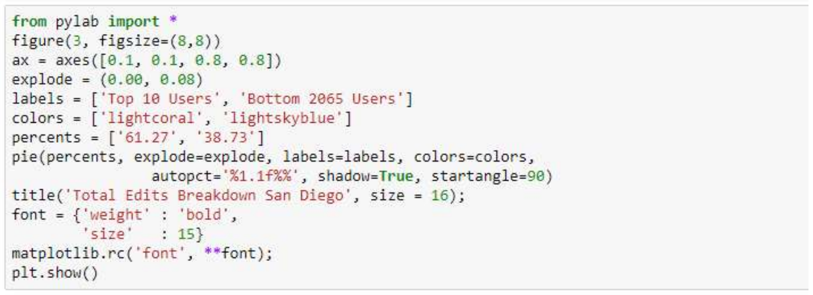


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

top10_df = pd.DataFrame(cur.execute(qtop10).fetchall())
top10_df.rename(columns={0: 'User', 1: 'Edits'}, inplace=True)
labels = list(top10_df['User'])

fig, ax = plt.subplots(figsize=(6,6))
ind = np.arange(10)
ax.bar(ind, top10_df['Edits'], width =0.8, color='blue', edgecolor='black', linewidth=0.8)
ax.set_xticks(ind)
ax.set_xticklabels(labels, rotation = 80, size = 16)
plt.title('Edits by Top Ten Users', size = 10); plt.ylabel('Edits', size = 15);
plt.show()
```

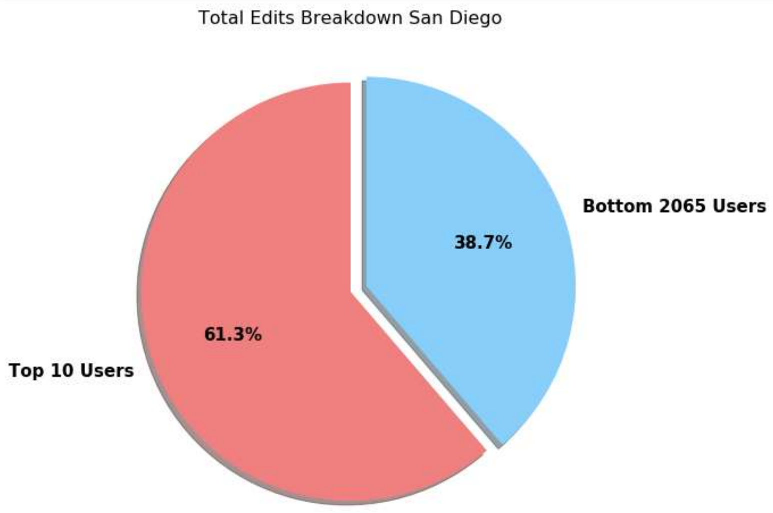


2. What was the breakdown of edits by users? I expected to see a view users making up the vast majority of edits.

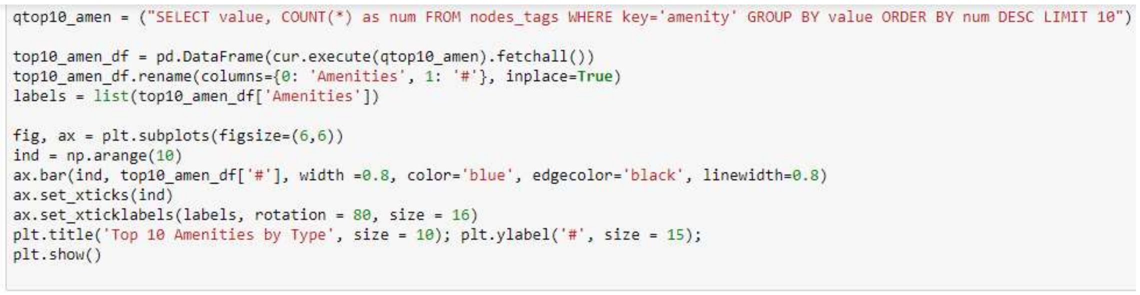
In order to answer that question, I first determined the total number of edits. I then aggregated the number of edits done by the top 10 users. Subsequently, I graphed the percentages in the pie chart below for a better visualization. It turns out that the top 10 users are contributing 61.3% of the edits while the other 2065 users are contributing only 38.7% of edits. This means that my hypothesis was true. In fact, a small number of users is responsible for the majority of edits.



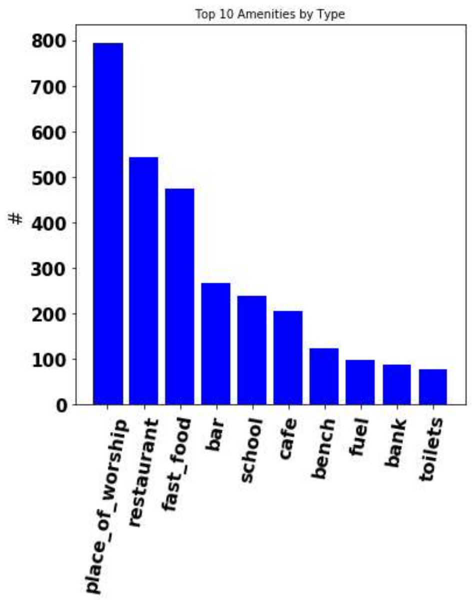
```
from pylab import *
figure(3, figsize=(8,8))
ax = axes([0.1, 0.1, 0.8, 0.8])
explode = (0.00, 0.08)
labels = ['Top 10 Users', 'Bottom 2065 Users']
colors = ['lightcoral', 'lightskyblue']
percents = ['61.27', '38.73']
pie(percents, explode=explode, labels=labels, colors=colors,
                autopct='%1.1f%%', shadow=True, startangle=90)
title('Total Edits Breakdown San Diego', size = 16);
font = {'weight' : 'bold',
        'size'   : 15}
matplotlib.rc('font', **font);
plt.show()
```



3. What are the most popular amenities by type?



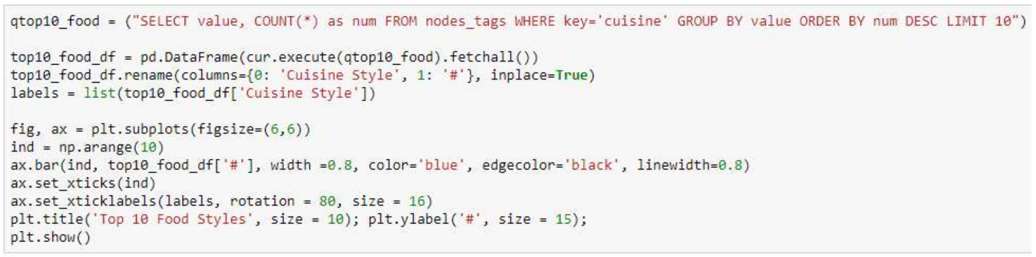
```
qtop10_amen = ("SELECT value, COUNT(*) as num FROM nodes_tags WHERE key='amenity' GROUP BY value ORDER BY num DESC LIMIT 10")

top10_amen_df = pd.DataFrame(cur.execute(qtop10_amen).fetchall())
top10_amen_df.rename(columns={0: 'Amenities', 1: '#'}, inplace=True)
labels = list(top10_amen_df['Amenities'])

fig, ax = plt.subplots(figsize=(6,6))
ind = np.arange(10)
ax.bar(ind, top10_amen_df['#'], width =0.8, color='blue', edgecolor='black', linewidth=0.8)
ax.set_xticks(ind)
ax.set_xticklabels(labels, rotation = 80, size = 16)
plt.title('Top 10 Amenities by Type', size = 10); plt.ylabel('#', size = 15);
plt.show()
```



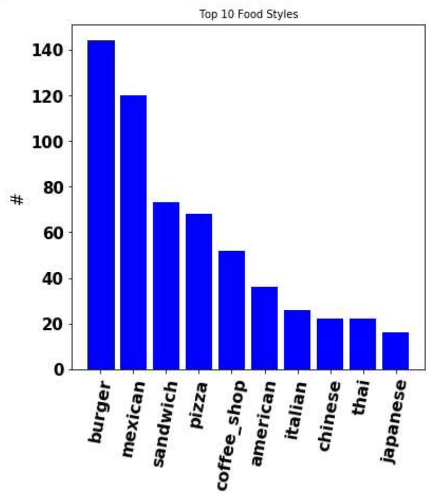
As it turns out the most popular amenities are “place of worhship”, “restaurant” and “fast food”.

4. What are the top 10 food styles?

Since two out of the top 3 amenities were related to food, I decided to look at the top 10 food styles and visualized them.



```
qtop10_food = ("SELECT value, COUNT(*) as num FROM nodes_tags WHERE key='cuisine' GROUP BY value ORDER BY num DESC LIMIT 10")

top10_food_df = pd.DataFrame(cur.execute(qtop10_food).fetchall())
top10_food_df.rename(columns={0: 'Cuisine Style', 1: '#'}, inplace=True)
labels = list(top10_food_df['Cuisine Style'])

fig, ax = plt.subplots(figsize=(6,6))
ind = np.arange(10)
ax.bar(ind, top10_food_df['#'], width =0.8, color='blue', edgecolor='black', linewidth=0.8)
ax.set_xticks(ind)
ax.set_xticklabels(labels, rotation = 80, size = 16)
plt.title('Top 10 Food Styles', size = 10); plt.ylabel('#', size = 15);
plt.show()
```



As we see above, the most popular food style is burger, followed by Mexican and sandwich – closely followed by pizza.
This pretty much fits, what I was seeing when was walking through the streets of Pacific Beach. I went to a lot of Burger and Mexican places. Delicious food 😊