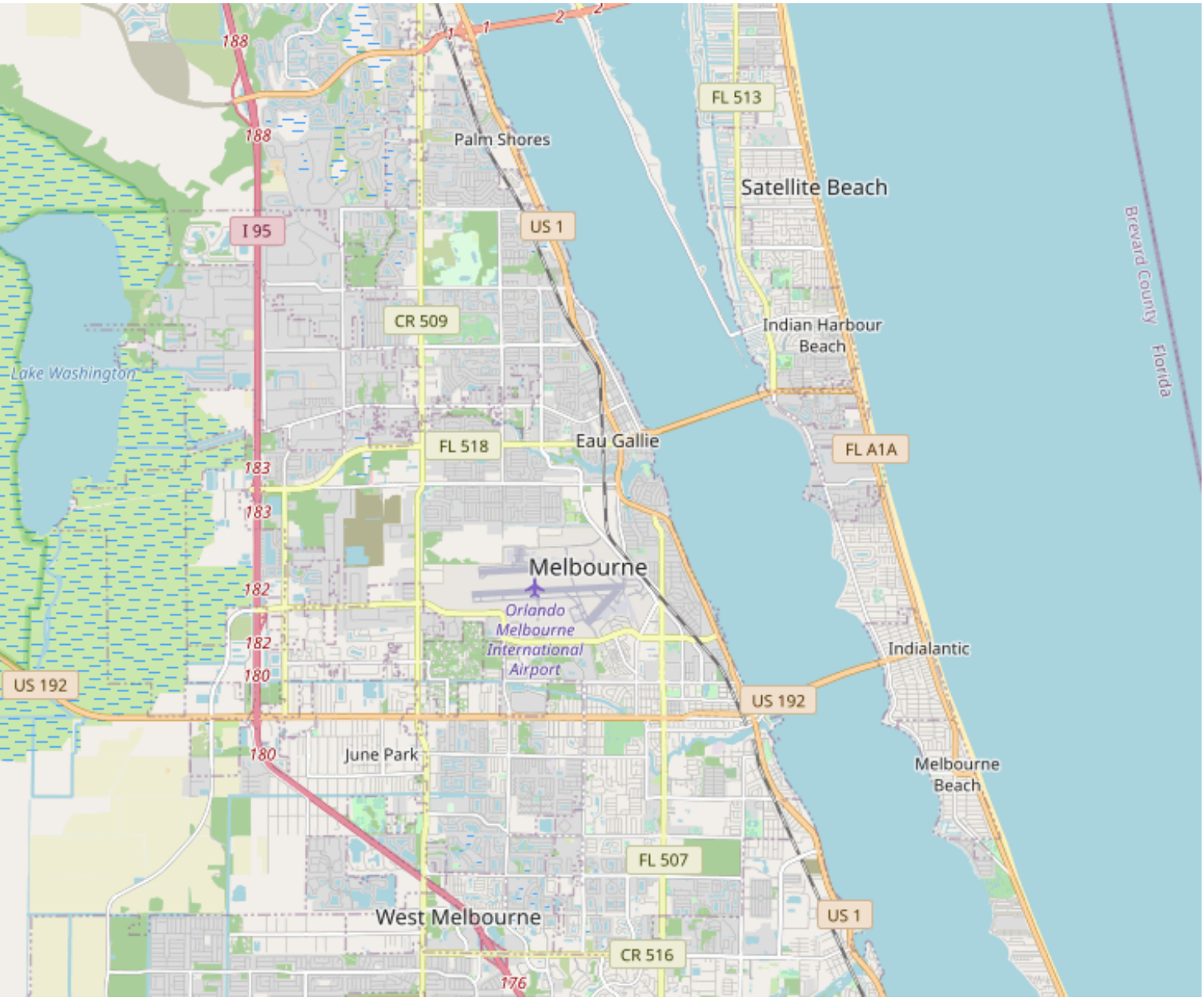
OpenStreetMap Project: Data Wrangling with MongoDB:

By: **Mohammed Bougayou**



Map Area: Melbourne, Florida

Located Melbourne, Florida from the map :
https://www.openstreetmap.org/relation/117646
OMS FIle : https://www.openstreetmap.org/export#map=12/28.1176/-80.6613

OpenStreetMap:

OpenStreetMap (**OSM**) is a collaborative project to create a free editable map of the world. The geodata underlying the map is considered the primary output of the project. The creation and growth of OSM has been motivated by restrictions on use or availability of map data across much of the world, and the advent of inexpensive portable satellite navigation devices.
#https://en.wikipedia.org/wiki/OpenStreetMap

Project Summary:

audit, update and normalize the dataset before putting it into a database, using python, audit techniques to iterate over the XML.OSM and perform certain activities.Then write out a JSON file of the cleaned data, and import the JSON file to MongoDB Database. Finally make queries to provide statistical information and numbers related to it.

OSM Dataset:

the main reason I picked Melbourne, Florida because it's my hometown, and I was just curious to look for information related to it. And get to know more about my city using MongoDB queries.

Files Sizes:

```
The downloaded file is 123.241036 MB

The json file is 211.020552 MB
```

Encounter problems:

before start and work on the OSM file, I reviews and investigate a sample set of data. Data entered is based on individuals user entries, therefore part of the Street names type are abbreviated. I update the variable 'Mapping to reflect the changes needed to fix the unexpected street types to the appropriate ones in the expected list.
I had some challenges with MongoDB loading on my mac, and for some reason the DB server disconnect. also python version when i process some queries learned in lesson i get an Error message.

I started first by counting how many tags in our XML file:

the task to use iterative parsing to process the sample file, my case input file and find out the tags names, and the count of tags. the function Count_tags return a dictionary with the tag name as key and number of times this tag can be encountered in the map as value.

##https://classroom.udacity.com/courses/ud032/lessons/768058569/concepts/8443086480923

```
    #Reads XML file: sample.xml and count each XML tag within the document

    for _, elemt in ET.iterparse(inputfile):
        tag = elemt.tag
        if tag not in tags:
            tags[tag] = 1
        else:
            tags[tag] += 1
    #returns a Dictionary consist of tag_names:(counts as values)

    return tags

pprint.pprint(count_tags(inputfile))
```

```
{'bounds': 1,
 'member': 23942,
 'meta': 1,
 'nd': 668786,
 'node': 556381,
 'note': 1,
 'osm': 1,
 'relation': 569,
 'tag': 210698,
 'way': 65070}
```

Identifying unique users contribute to the map area: Melbourne FL:

we are going to explore the data and find out how many unique users made changes to the map in our area. sea re going to define the function Process_map to build a set of userid's in XML. then we are using len() to present the number of unique users.

```
#identifying unique users contribute the to the map area: Melbouren, FL.
"""
Your task is to explore the data a bit more.
The first task is a fun one - find out how many unique users
have contributed to the map in this particular area!

The function process_map should return a set of unique user IDs ("uid")
"""

def process_map(inputfile):
    users = set()
    for _, element in ET.iterparse(inputfile):
        for e in element:
            if 'uid' in e.attrib:
                users.add(e.attrib['uid'])

    return users
#the function process_map return a set of unique user "uid"
users = process_map(inputfile)
print('Number of unique users:', len(users))
```

```
Number of unique users: 797
```

Checking the "K" value for each "tag" for potential problems:

we built three regular expressions: lower, lower_colon, prob_chars:

-**Lower:** matches strings containing lower case characters.

-**Lower_colon:** matches strings containing lower case characters and a single colon within the string

-**Probl_chars:** matches characters that cannot be used within keys in MongoDB.

```
import re

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

def key_type(element, keys):
    #tag's attribute matches a regular express and counts # tags.
    if element.tag == "tag":
        if prob_chars.search(element.attrib['k']):
            keys['prob_chars'] +=1
            #print element.attrib['k']
        elif lower_colon.search(element.attrib['k']):
            keys['lower_colon'] +=1
        elif lower_case.search(element.attrib['k']):
            keys['lower_case'] +=1
        else:
            keys['other'] +=1


    return keys


def process_map(inputfile):
    keys = {"lower_case": 0, "lower_colon": 0, "prob_chars": 0, "other": 0}
    # Iterates through an XML file and create a Dict of keys/count.

    for _, element in ET.iterparse(inputfile):
        keys = key_type(element, keys)

    return keys

keys = process_map(inputfile)
pprint.pprint(keys)
```

```
{'lower_case': 140764, 'lower_colon': 66274, 'other': 3660, 'prob_chars': 0}
```

Audit, Update street names:
the update_name function process a string with street name as an argument and return a full complete street type.

```
def update_name(name, mapping):
    m = street_type_re.search(name)
    street_type = m.group()
    if street_type not in expected:
        if street_type in mapping.keys():
            new_street_type = mapping[street_type]
            name = name.replace(street_type, new_street_type)

    return name
```

- after completing the function element that parse.we used import os to get the size of the downloaded file and json file.

```
import os
print('The downloaded file is {} MB'.format(os.path.getsize(inputfile)/1.0e6))
```

```
The downloaded file is 123.241036 MB
```

```
print('The json file is {} MB'.format(os.path.getsize(inputfile + ".json")/1.0e6))
```

```
The json file is 211.020552 MB
```

we processed the file using openstreetmap.py code, result with json file output. Now we are going to import to json file to MongoDB using the command line. By the way it does take couple minute to load.

mongoimport –db openstreemap –collection melbourne_florida –drop –file ~/Downloads/sample.json

#https://docs.mongodb.com/guides/server/import/

Data queries:

Number of documents:

```
>db = client["openstreetmap"]
melbourne_florida = db[“melbourne_florida"]
print(melbourne_florida.find().count())
```

#https://classroom.udacity.com/courses/ud032/lessons/745498943/concepts/7347306510923

Result = 621451

GovernmentType:

>I want to find document that do have a govermentType field: I was surprised that they were None. result is 0.

#https://classroom.udacity.com/courses/ud032/lessons/745498943/concepts/7347306470923

#i want to find document that do have a govermentType field.

```
db.melbourne_florida.find({"governmentType" : {"$exists" : 1}}).count()
```

Result = 0

Unique users that contribute to the data:

>len(melbourne_florida.distinct('created.user'))

Result = 778

Number of nodes:

>db.melbourne_florida.find({“type”:{"$in":['node']}}).count()

Result = 556381

Number os ways:

>db.melbourne_florida.find({“type”:{"$in":['way']}}).count()
————-65070————

List of the main contributors to our dataset:

>cursor= db.melbourne_florida.aggregate([{"$group":{"_id":"$created.user", "count": {"$sum":1}}}, {"$sort":{"count":-1}}, {"$limit":10}])

for document in cursor:

    print(document)

```
{'_id': 'Ian Swire', 'count': 111093}
{'_id': 'floridaeditor', 'count': 66009}
{'_id': 'Panther37', 'count': 62191}
{'_id': 'NE2', 'count': 61076}
{'_id': 'chachafish', 'count': 32837}
{'_id': 'mahahahaneapneap', 'count': 25559}
{'_id': 'LeTopographeFou', 'count': 22796}
{'_id': 'grouper', 'count': 21930}
{'_id': 'Easky30', 'count': 19119}
{'_id': 'GlittrGrl', 'count': 12856}
```

Now i will querying and get information from the data regarding amenities and count, also want to know how many bars are in town.

```
#count of bars in melbourne based of the match.
results = db.melbourne_florida.aggregate([{'$match': {'amenity': 'bar'}}])
Very surprised that there is only 12 bars, I thought lot more than that.
#count for different types of amenities. only the top 10.
results = db.melbourne_florida.aggregate([{"$group":{"_id":"$amenity", "count":
{"$sum":1}}}, {"$sort":{"count": -1}},{'$limit': 11}])
for amenity in results:
    print(amenity)
```

```
{'_id': None, 'count': 619100}
{'_id': 'parking', 'count': 893}
{'_id': 'restaurant', 'count': 186}
{'_id': 'fountain', 'count': 150}
{'_id': 'fast_food', 'count': 142}
{'_id': 'place_of_worship', 'count': 131}
{'_id': 'school', 'count': 110}
{'_id': 'fuel', 'count': 96}
{'_id': 'shelter', 'count': 72}
{'_id': 'bank', 'count': 50}
```

very interesting to collect this returns, and how you can use it in your favor. we can say that there is more restaurant than fast food places. Florida is state of hurricanes, very good to know that they're 72 shelters available.

Zip_codes in town:

```
results = db.melbourne_florida.aggregate([{'$match': {'address.postcode':
{'$exists': 1}}},
                          {'$group': {'_id': '$address.postcode',
                                      'count': {'$sum': 1}}},
                          {'$sort': {'count': -1}}])
for zipcode in results:
    print(zipcode)
```

```
{'_id': '32940', 'count': 212}
{'_id': '32901', 'count': 165}
{'_id': '32907', 'count': 143}
{'_id': '32935', 'count': 81}
{'_id': '32909', 'count': 69}
{'_id': '32904', 'count': 44}
{'_id': '32937', 'count': 23}
{'_id': '32934', 'count': 21}
{'_id': '32950', 'count': 21}
{'_id': '32955', 'count': 13}
{'_id': '32903', 'count': 13}
```

i can conclude that zip_codes entered cleaned and don't need any modification or audit.

Conclusion :

lot information that can we produced from the dataset and helpful information that can we used. I found MongoDb are lot simpler and easier to utilize. Accuracy of the data is very crucial once you have it normalized and cleaned, lot information is added by users and sometimes needs more analyzing and investigation. This project is an eye opening to the future and how we can use data in our favor.