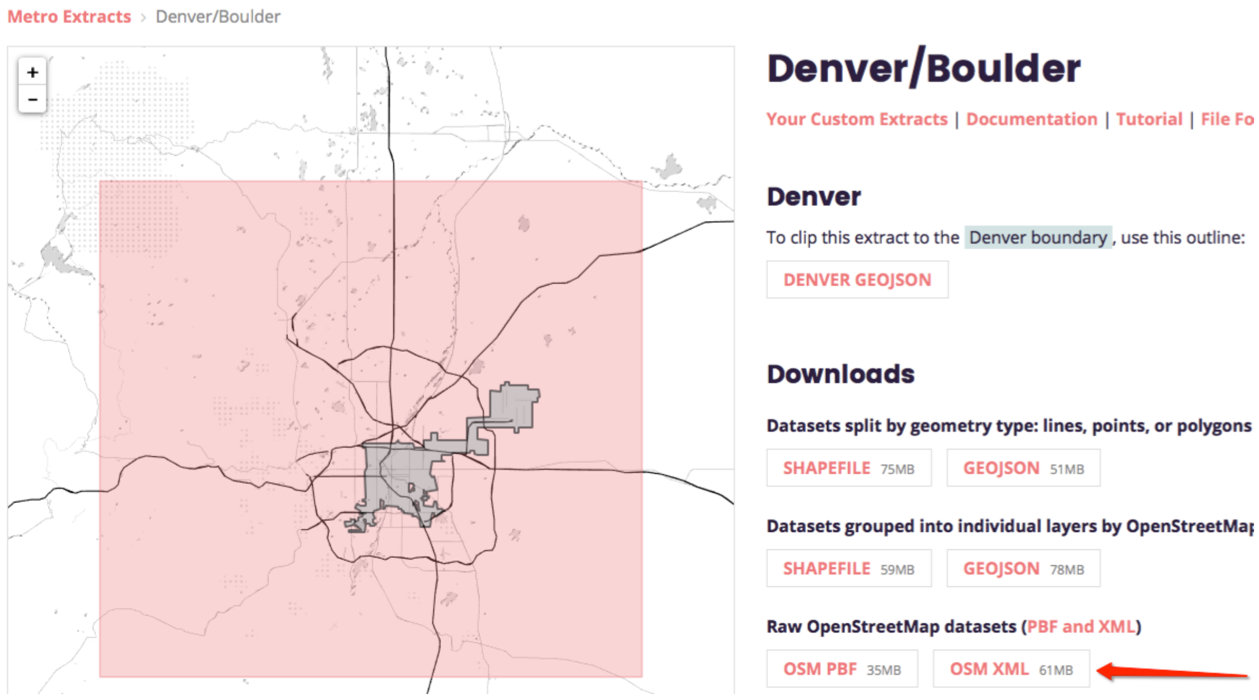
OPEN STREET MAP DATA WRANGLING WITH SQL

Awad Bin-Jawed

I will demonstrate the process of data wrangling for the city of **Denver, CO, USA**. This project will be an iterative process, so hopefully I will get a desired result as accurate as possible after an extensive cycle of repeating operations. This is why we say this whole cycle is convergent.

I extracted the OSM XML file of Denver.

I will need to audit the street types and standardize the data; this way, I will know exactly what to extract.
Mapzen URL https://mapzen.com/data/metro-extracts/metro/denver- boulder_colorado/85928879/Denver/


Metro Extracts › Denver/Boulder
+
-
Denver/Boulder
Your Custom Extracts | Documentation | Tutorial | File Fo
Denver
To clip this extract to the Denver boundary, use this outline:
DENVER GEOJSON
Downloads
Datasets split by geometry type: lines, points, or polygons
SHAPEFILE 75MB
GEOJSON 51MB
Datasets grouped into individual layers by OpenStreetMa
SHAPEFILE 59MB
GEOJSON 78MB
Raw OpenStreetMap datasets (PBF and XML)
OSM PBF 35MB
OSM XML 61MB

DATA AUDITING

The **audit.py** script generates two outputs:

1) **audit.txt**
2) **audit_xsd.txt**

The **audit.txt** contains captured output of 6 functions:

```
print(str(audit_street(OSM_FILE)))
fix_street(audit_street(OSM_FILE))
print(str(audit_city(OSM_FILE)))
print(str(audit_postcode(OSM_FILE)))
output.write(str(audit_phone(OSM_FILE)))
fix_phones(phone_list)
```

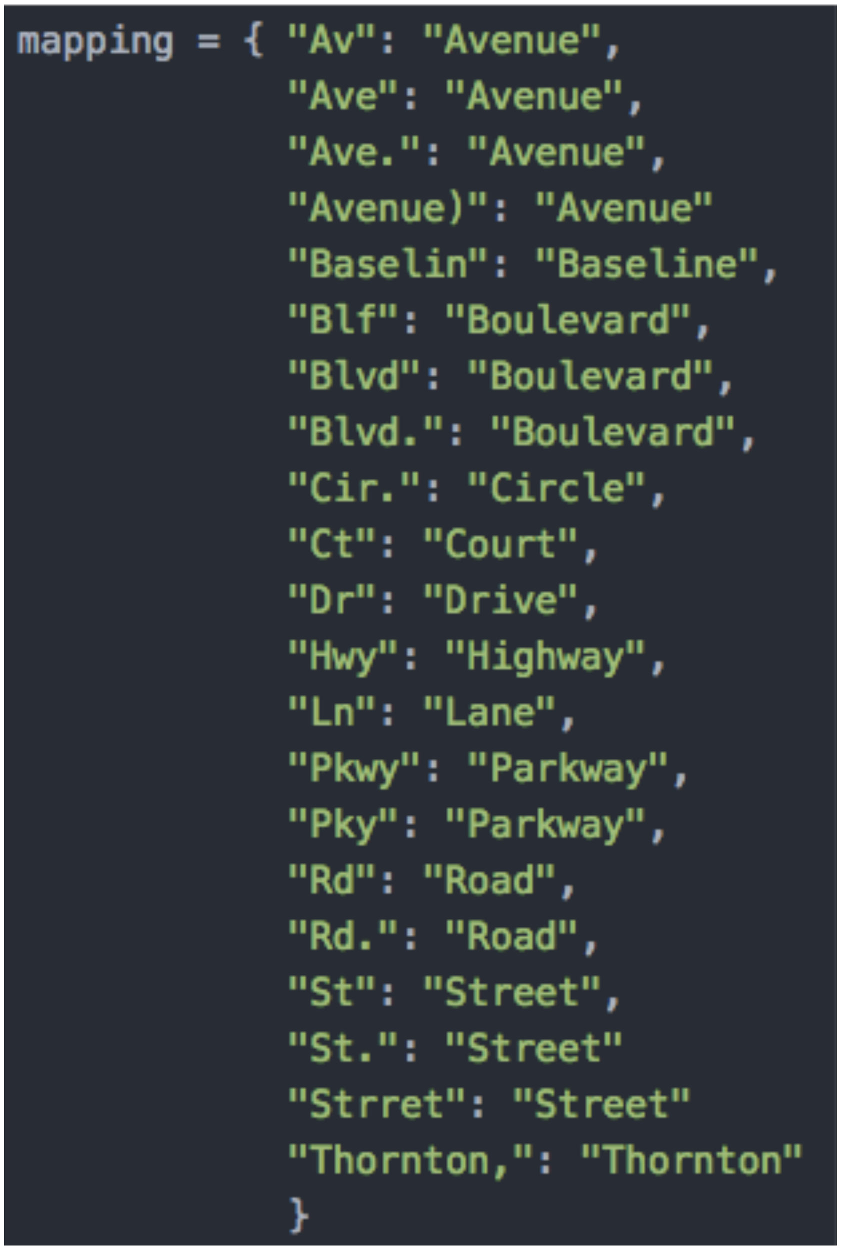
The **audit_xsd.txt** contains the result of validation of the full map set.

Misspelled/Abbreviated variables:

I added “What is Expected” and updated the inconsistencies:

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

In addition, I added the mapping:

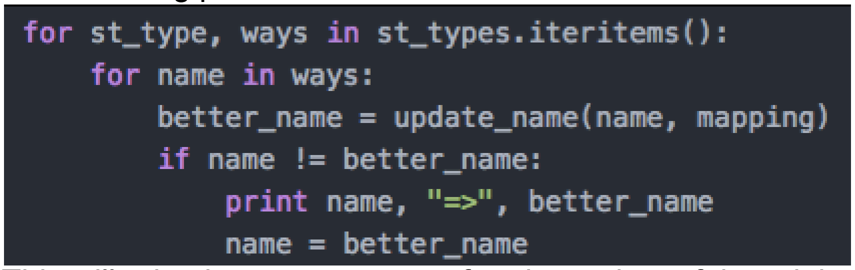


```
mapping = { "Av": "Avenue",
            "Ave": "Avenue",
            "Ave.": "Avenue",
            "Avenue)": "Avenue"
            "Baselin": "Baseline",
            "Blf": "Boulevard",
            "Blvd": "Boulevard",
            "Blvd.": "Boulevard",
            "Cir.": "Circle",
            "Ct": "Court",
            "Dr": "Drive",
            "Hwy": "Highway",
            "Ln": "Lane",
            "Pkwy": "Parkway",
            "Pky": "Parkway",
            "Rd": "Road",
            "Rd.": "Road",
            "St": "Street",
            "St.": "Street"
            "Strret": "Street"
            "Thornton,": "Thornton"
            }
```

Abbreviations:

Av needs to be converted to Avenue
Ave needs to be converted to Avenue
Ave. needs to be converted to Avenue
Avenue) needs to be converted to have the right parenthesis “)” removed
Baselin needs to be converted to Baseline
Blf needs to be converted to Boulevard
Blvd needs to be converted to Boulevard
Blvd. needs to be converted to Boulevard
Cir. needs to be converted to Circle
Ct needs to be converted to Court
Dr needs to be converted to Drive
Hwy needs to be converted to Highway
Ln needs to be converted to Lane
Pkwy needs to be converted to Parkway
Pky needs to be converted to Parkway
Rd needs to be converted to Road
Rd. needs to be converted to be Road
St needs to be converted to Street
St. needs to be converted to Street
Strret needs to be converted Street

The following print statement:



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

This will print the street names after the update of the original, unclean street names.
Essentially, the goal of the scripts is to analyze the data of Denver.
Using the editor (in this case Atom), you may manually analyze the large .osm file.
There is a method run() in all files which can be called from report.py
or in an interpreter.

The function of shape_element() in **data.py** is it converts a first level **<node>** or **<way>** element from a OSM XML file into a dictionary.

It also tests second level **<tag>** elements on problematic characters and converts second level **<tag>** or **<node>** elements into subdictionaries.

If the element tag is "node", it returns a dictionary in the format {"node": {'id': 1234, ...}, "node_tags": [{'id': 4321, ...}, ...] }.

If the element tag is "way", it returns a dictionary in the format {"way": {'id': 1234, ...}, "way_nodes": [{'id': 4321, ...}, ...], "way_tags": [{'id': 1423,...}, ...] }.

The line in **data.py** that can be replaced….

```
###
def run():
SAMPLE_PATH = OSM_PATH[0:-4] + '_sample.osm'
process_map(SAMPLE_PATH, validate=True)
#run()
###
```

with a common block:

```
if __name__ == “___main___”:
SAMPLE_PATH = OSM_PATH[0:-4] + '_sample.osm'
process_map(SAMPLE_PATH, validate=True)
```

But in my experience, I found the data would not be better processed. If I wish to process larger OSM_PATH, I can write process_map(OSM_PATH, validate=False). The data is valid in this sense, but the validation in this script is very slow.

mapparser.py
This script will yield a defaultdic dictionary of tags and occurrences.

users.py
This script will yield a defaultdic dictionary of user’s id numbers.

tagtype.py
This script will yield a simple dictionary of categorized <tag> tags. It also outputs problematic tags.

After auditing, the next step is to prepare the data for insertion to a SQL database. We parse the elements in the OSM XML file, which will transform them from .doc format to tabular format. The process involved using the iterparse through the elements in the XML and shaping the elements into several structures. The schema was to ensure the data is in the correct format. Next we attempt to write each data structure to the appropriate .csv files.

DATA OVERVIEW

File Sizes:

```
denver-boulder_colorado.osm ........................  68  MB
nodes.csv ...........................................  4.1 MB
nodes_tags.csv: ..................................... .171 MB
ways.csv: ........................................... .327 MB
ways_nodes.csv: .....................................  1.3 MB
ways_tags.csv: ...................................... .659 MB
```

Upon executing **query.py**, on the respective .csv files, we are able to output statistical overview of the dataset:

```
Number of nodes ..................................  46827
Number of nodes_tags .............................  4628
Number of ways: ..................................  5189
Number of ways_nodes: ............................  52491
Number of ways_tags: .............................  18812

Number of unique users: ..........................  853
Number of cafes ..................................  5
Number of restaurants ............................  28
```

Problems Encountered:

sample.py

This script had a broken format:

```
<?xml version="1.0" encoding="UTF-8"?>
<osm>
        <node changeset="7551724" ... />
                <node />
....
                <tag k="type" v="restriction" />
        </relation>
        </osm>
```

This created problems, which were due to output.write() breaking indentations.

The first one can be fixed by output.write('<osm>\n\t'), the second one can be fixed by manually editing the last line in the new file.

data.py

This program creates 4 csv files from denver-boulder_colorado_sample.osm

to be added to the database. The validator in it WILL NOT raise errors because

it doesn't check the nested tags.

The xml+cerberus+schema code is very slow. It is much slower than the lxml+.xsd code.

Additional Ideas

The state of Colorado has recently been grabbing headlines due to a controversial legislative action... the legalization of recreational marijuana in 2012. The Colorado Amendment 64 was a successful measure to overrule the longstanding drug policy of Colorado’s own constitution, which has kept prohibited cannabis since 1917.
It would be interesting to map out cannabis and crime in the state of Colorado. More specifically, the Denver metro area.

The crime CSV file, extracted from the Denver Open Data Catalog comprised of data sets of criminal offenses during the past 5 years (since 2012 until now). See **mapcrime.py** in the repositories

The shops CSV file, extracted from the Open Colorado Data Sets, comprised a listing of active shops/dispensaries as of November 2017. See **mapshops.py** in repositories Google maps was utilized via pygmaps. I show the first 10 places. You can change the variable count (I use count=10) and get more places on the map.

Some Anticipated Problems:
1) A question that arises is, “How can we be sure if the crimes are drug-related charges?” If they are DUI, it would require further inquiry to distinguish between cannabis or other intoxicants (ie, alcohol).

2) The Crime and Shops datasets are downloaded from a website. This means they are merely snapshots at the time of download.

Some Possible Solutions:
1) A possible solution would be to filter out the crimes to those specifically cannabis-related. This would eliminate crimes otherwise unrelated to cannabis. This would allow us to map out the crimes in a less subjective manner.
2) The Crime and Shops would need a clean install and replacement for each analysis. This would ensure the datasets are recent and up-to-date.

For a map comparing both crime and shops, See **MapBothCrimeAndShops.py** in repositories.

I show the first 10 places. You can change the variable count (I use count=10) and get more places on the map.

For a map comparing both crime and shops, see **MapBothCrimeAndShops.py** in repositories.