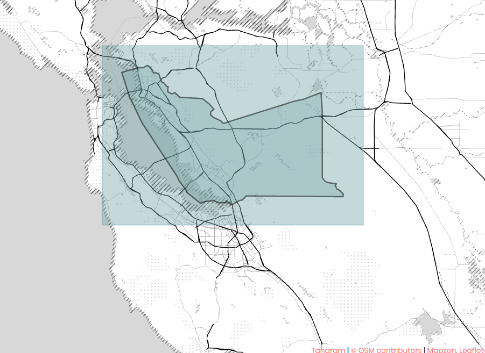
# Project: Wrangle OpenStreetMap Data

Prepared by: W. McCracken

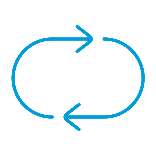
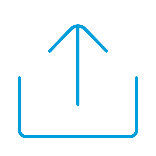
Alameda County Extract

The purpose of this project is to demonstrate that I am capable of:

* Assessing the quality of a data set for validity, accuracy, completeness, consistency and uniformity
* Parsing and gathering data from a popular file formats such as .csv, .json, .xml, and .html
* Processing data from multiple files or very large files that can be cleaned programmatically.
* Learning how to store, query, and aggregate data using SQL.

I chose the area of Alameda County shown in the shaded box above as Udacity is headquartered and I live there. Reference: Alameda County Map via mapzen - <https://mapzen.com/data/metro-extracts/your-extracts/11184ceaf7a2> Reference to download of the OSM file: <https://s3.amazonaws.com/mapzen.odes/ex_mj5W3UGssTC68RBcyHym7dM1SuvcF.osm.bz2> The boundaries of the map are: minimum latitude ="37.3900729" , minimum longitude ="-122.4546969" , maximum latitude ="37.9700843", maximum longitude="-121.3876639". The process overview below shows the high level flow of the activity to complete the project:

**Process Overview:**

Transform HTML to CSV

Sample and explore data

Scrape and clean data

Export to CSV

Import to SQL

Query/Analyze/Report

Download data

**Sampling:** I ran a quick count tags on the full data full data set and took a few minutes to process even though I have 8GB of RAM and processing the 1.5GB file should not be a problem. I used the sample extract code (sampleOSMfilecreation.py) and extracted three samples, every 10, every 20 and every 30 nodes. Lower, lower colon, other and problem character columns are a count of the format fields found in each data set. Skipping 30 nodes still created a large file so I opted to skip every 95 nodes to develop test code for testing. If the table below; nodes, node tags, ways, ways nodes and ways tags are a count of the number of records in each csv file. A comparison of the files follow:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Data set | Lower | Lower colon | Other | Problem Characters | Nodes | Node Tags | Ways | Ways Nodes | Ways Tags |
| Full | 1,66,0897 | 781,175 | 32,877 | 91 |  |  |  |  |  |
| Sample every 10th | 165,860 | 78,142 | 3,280 | 16 | 724,758 | 30,108 | 90,272 | 854,237 | 214,409 |
| Sample every 20th | 83,174 | 39,113 | 1,650 | 6 | 362,378 | 15,481 | 45,136 | 426,568 | 107,163 |
| Sample every 30th | 55,236 | 25,813 | 1,058 | 8 | 241,587 | 10,032 | 30,091 | 285,633 | 71,132 |
| Sample every 95th | 17,470 | 8,351 | 336 | 7 | 76,441 | 3,152 | 9,502 | 88,434 | 22,708 |

Since sampling every 95th element creates files with a good variety, I used this to developed cleaning and analysis algorithms before applying this to the full file. To get a sense for the types of elements in each of the files I ran the code to prepare the data for analysis and created csv files for each sample map. I then opened them to explore them in Microsoft Excel by selecting a data range and applying a filter to the fields. This is a very fast way to explore variation and range in data before applying more sophisticated/comprehensive Python techniques. As a result, the 10th sample is too large to open in excel so I studied the fields in each of the 20th files to get a sense for the values in each file.

**Exploring:**  After exploring the data in excel, I created to exploratory python scripts count of the values of child tags to dig deeper into the data and find fields that might need cleaning. Exploring the values of most common “child” (v) tags (exploretypes.py) such as street, postcode, house number, cuisine indicated very few issues worth cleaning. The phone numbers were in several different format warranting cleaning. It appeared the data was fairy clean so I raised the review to look at the value “parent” (k) tags. There are 452 parent “tags” in the sample. (explore\_k\_tagtypes.py) A review of the parent tags helped identify a few fields that could be cleaned such as the state, country and variations of the name field. In addition, this review demonstrated that the value of one of the most widely used tags in the map file “name” had variations in the tag that warrant cleaning.

The basic function in both the explore python scripts is a function “audit\_element\_type” shown below:

‘’’

def audit\_element\_type(element\_types, element\_name):

m = element\_type\_re.search(element\_name)

if m:

element\_type = element\_name

element\_types[element\_type] += 1

def is\_element\_name(elem):

return (elem.tag == "tag") and (elem.attrib['k'] == "phone")

‘’’

This loops though the osm file and adds the elements being inspected to a dictionary with a counter. A separate function prints the dictionary so you can see the results immediately.

***Example 1:*** Changing the element name in the “is\_element\_name” function in the exploretypes.py script to “phone” generated the following:

‘’’ Sample Output

(650) 961-8600: 1

(800) 275-8777: 1

+1 (415) 641-9885: 1

+1 (510) 524-2532: 1

+1 415 621 0874: 1

+1 415 800 7416: 1

+1 510 4440919: 1

+1 510 452-1499: 1

+1 510 545 4356: 1

+1 510-655-6336: 1

+1-415-932-6531: 1

1-415-241-6380: 1

1-650-965-3189: 1

415-648-4157: 1

‘’’

This shows the variety of ways phone numbers are included in the data which led to adding a cleaning function for this field to the code to prepare the data for the database.

***Example 2*** : Changing the element name in the “is\_element\_name” function to “addr:street” generated the following :

‘’’ Sample Output

CA: 127

Ca: 2

ca: 1

California: 1

‘’’

This shows the four ways state is included in the data leading to adding a cleaning function for this field to the code to prepare the data for the database.

I applied the same exploratory code to elements such as “addr:postcode”, “addr:street”, “amenity” and several others. While there was some variation in some of the elements I explored, I chose to focus on cleaning three “child” value fields: “phone”, “addr:state” and “addr:country”. Cleaning algorithms for phone number, state and country will be elaborated on in the cleaning section below.

***Explore “k” types:*** The second exploratory script, explore\_k\_tagtypes.py is almost identical to the exploretypes.py with the exception of the is\_element\_name function. In the “k-type” script we are just checking for the count of all the various tags. This lets you know which tags might need to be cleaned.

‘’’

def is\_element\_name(elem):

return (elem.tag == "tag")

‘’’

***Example 3*** : Executing the , explore\_k\_tagtypes.py generated the following view of the “name” tag:

‘’’ Sample Output

name: 1391

name:en: 4

name:prefix: 5

name:vi: 1

name:zh: 4

name\_1: 26

name\_2: 1

‘’’

Note: Exploration is an iterative process. Exploring the values of name:vi and name:zh with the exploretypes.py script uncovered chines characters in the child elements values for these

Given that the name tag is the most prominent tag in the file, I developed a cleaning algorithm to standardize the name tag. ”. The cleaning algorithm for the tag “name” will be elaborated on in the cleaning section below. Note, I developed a cleaning algorithm for a “tag” value so that it could be easily used to clean any “tag” value in this or any other xml file. It will be useful in the future.

***Other auditing / exploratory notes:***

* Auditing Validity – Initial attempts to scrape the data failed as a userid was expected for some records and it was not present. An additional step was added to the parsing function to skip records with no userid. The numeric range for postal codes is in a tight range all beginning with 94.
* Auditing Accuracy – A review of the accuracy of a few fields indicated relatively accurate data. For example, four counties appeared in the data which are all valid counties in the area. While the country identified is all United States, four abbreviations were found which led to the cleaning step discussed above.
* Auditing Uniformity – I validated several fields for uniformity using excel filtering techniques discussed above. For example, all of the postcode values began with “94” and all the latitude and longitude values were numeric and within the minimum and maximum longitudes and latitudes of the map.

**Scraping/Cleaning and Export to CSV:**  The main script for the project reads the OSM file (XML format) and parses though it to create five CSV files that can be loaded into SQL for analysis. (prepfordatabase.py) I added a function (def update\_name) to standardize the state and country. This function could be modified to clean any “child” element. I added a function (fixphone) to standardize the phone number. To simplify the code, I modified added an additional parameter to the def update\_name function which allows for standardization of either a parent “k” level label or a child “v” value.

**Import to SQL:**  I incorporated use of the SQLalchemy package due to the large size of the files and performance objectives. This simplifies the code to import data to a virtual SQL database and increased the speed of queries of the data. The SQL queries are identical to traditional SQL queries with the exception that they do not have to be wrapped into a “cursor” function. After reading the project csv files into pandas dataframes using the pd.read\_csv command, the following was used to import the dataframes into SQL tables:

‘’’

eng = create\_engine("sqlite://", encoding='utf8')

conn = eng.connect()

NODES.to\_sql('nodes', conn)

NODE\_TAGS.to\_sql('node\_tags',conn)

WAYS.to\_sql('ways',conn)

WAYS\_NODES.to\_sql('ways\_nodes',conn)

WAYS\_TAGS.to\_sql('ways\_tags',conn)

‘’’

Note: I had difficulty importing data into SQL with the basic SQLalchemy format command. After hours of investigation and some guidance from my mentor, I determined that the cause of the issue was Chinese characters in the values of some “value” tags. I contemplated excluding Chinese characters but since these are legitimate names and descriptions of businesses in this area, I added an “encoding='utf-8” parameter to the read\_csv and creation of the sqlite engine connection to accept Chinese characters. This solved the problem.

**SQL Queries:**  I created a variety of SQL as follows:

‘’’

###**Record Count Metric database queries**

numberofnodes = pd.read\_sql("SELECT COUNT(\*) FROM nodes",eng)

numberofnode\_tags = pd.read\_sql("SELECT COUNT(\*) FROM node\_tags",eng)

numberofways = pd.read\_sql("SELECT COUNT(\*) FROM ways",eng)

numberofways\_tags = pd.read\_sql("SELECT COUNT(\*) FROM ways\_tags",eng)

numberofways\_nodes = pd.read\_sql("SELECT COUNT(\*) FROM ways\_nodes",eng)

‘’’

Results – edited for ease of reading

|  |  |
| --- | --- |
| Table\_name | Records |
| Nodes | 76291 |
| Node Tags | 3152 |
| Ways | 9502 |
| Ways\_Tags | 22708 |
| Ways\_Nodes | 88434 |

#**Distinct Users**

**‘’’**

distinct\_users = pd.read\_sql("SELECT COUNT(DISTINCT nodes.uid) as distinctusers \

FROM nodes JOIN ways on ways.uid = nodes.uid",eng)

print "Distinct Users:", distinct\_users

‘’’

**Results: Distinct Users: 360**

**#Timestamps**

most\_recentnode = pd.read\_sql("SELECT DISTINCT timestamp \

FROM nodes Order by timestamp DESC Limit 2",eng)

print "Most Recent node updates:", most\_recentnode

oldestnode = pd.read\_sql("SELECT DISTINCT timestamp \

FROM nodes Order by timestamp Limit 2",eng)

print "Oldest node updates:", oldestnode

**Results: Time stamps**

*Most Recent node updates:*

2018-01-14T15:02:08Z

2018-01-14T15:02:07Z

*Oldest node updates*:

2007-09-21T12:00:49Z

2007-09-25T10:40:11Z

**Conclusions and recommendations:** At the onset of this project I was skeptical of the overall value and practical application to the area of work I am in. Upon completion, I very pleased that I have built a framework for obtaining, exploring, cleaning and loading data into a SQL format for summary and analysis. I will be able to replicate for many simple applications I am involved in.

**Recommendation:** While OpenstreetMap is “open source” data, there should published standards for common fields like name tags and phone formats. This will improve the overall quality and usefulness of the data in applications.

**Fun Observation from the map:** “[Imagine your future! What do you want to learn today? What's your dream job?”](https://www.udacity.com/pathfinder) .....This appears to be a statement made by a school, university or learning institution. A quick comparison of the high level information about Stanford, University of California Berkeley and Udacity reveal very different profiles. (see table below) Like musical artist that release music in obscure genres, Udacity might consider listing itself as a “school” as opposed to “company” to improve search results.

Insert table here!!!!