Wrangling OpenStreetMap Data Project

Locale: Irvine, CA USA
https://en.wikipedia.org/wiki/Irvine%2C_California
https://www.openstreetmap.org/relation/114485#map=12/33.6868/-117.7734

I've chosen my city of residence, *Irvine CA* as the area of the world in which I'm interested. My goal was to wrangle a bounded box of map data comprising the *Irvine metroplex* (which includes small portions of neighboring cities: Tustin; Santa Ana; Newport Beach; Costa Mesa; Fountain Valley; & Lake Forest) from OpenStreetMap (OSM); audit a sample of map data; and load it into a SQL database using the schema provided for the project. Then, I'd identify how much data cleansing would be required. I would perform this process iteratively, using larger datasets for each iteration.

Sampling the Map Data
I downloaded two sample files of XML OSM data using the *Overpass API & Overpass Turbo remote*. *Turbo remote* was used to test and develop the XML queries. The *Overpass API Query Form* was used to perform the ML extraction and downloads, as shown below:

Small sample: (uncompressed file size 3.2MB)	Intermediate sample: (uncompressed file size 12.8MB)

[out:xml];
(
  // query
  node(33.6785,-117.8061,33.7045,-117.7704);
  way(33.6785,-117.8061,33.7045,-117.7704);
  relation(33.6785,-117.8061,33.7045,-117.7704);
);
out meta;
	[out:xml];
	(
	  // query
	  node(33.6655,-117.8239,33.7175,-117.7525);
	  way(33.6655,-117.8239,33.7175,-117.7525);
	  relation(33.6655,-117.8239,33.7175,-117.7525);
	);
	out meta;

Initial Wrangling Process Prep.
After downloading the above sample XML files, I performed a cursory inspection of the sample contents and didn't detect any glaring problems with the data. I ran a series of Python programs, derived from the course case study, using a Jupyter notebook to prepare and convert the XML data to .csv files. I then loaded each .csv file into a corresponding table created in the **OSM_Project** database for more detailed data auditing.

It appeared that the OSM contributors (users) had done a fairly good job of maintaining the OSM data in a consistent and uniform manner. Nevertheless, I noted a few peculiarities:

1. Some street names have designations such as *Avenue*, *Circle*, *Drive*, *Lane*, *Loop*, *Road, or Street*. However, many only have a primary name such as *Arborglen*, *Morning Dove, Waynesboro*. I opted not to alter this because such naming convention, for streets in Irvine, is customary.
2. It appeared that there were a few inconsistencies with city names, with street names and numbers, and with zip codes.

With the wrangling process working smoothly for the samples, and the apparently small number of detected data anomalies, I decided the best next steps would be to download the full set of data for the project and again use SQL to expose more data anomalies

The full dataset download was performed using the following *Overpass API Query*:

Full dataset: (file size 55.5MB)

```
[out:xml];
(
  // query
  node(33.6304,-117.8621,33.7495,-117.7039);
  way(33.6304,-117.8621,33.7495,-117.7039);
  relation(33.6304,-117.8621,33.7495,-117.7039);
);
out meta;
```

Auditing the Full Dataset

Accuracy & Completeness

Accuracy of the data is difficult to establish, since this requires the data to be compared to some 'gold' standard. Since this data includes my home address and my neighbors' addresses, I can at least verify that this subset of data conforms to my knowledge of our housing development. To this end, I ran the following SQL query to list my address (**highlighted**) and those of my **neighbors**.

```
SELECT street_name.id, house_no.id, street_name.key, house_no.key, street_name.value,
house_no.value
FROM nodes_tags as street_name, nodes_tags as house_no
WHERE street_name.key = "street"
and house_no.key = "housenumber"
and street_name.value = "Royal Victoria"
and street_name.id = house_no.id
ORDER BY CAST(house_no.value as integer);
```

3818125688|3818125688|street|housenumber|Royal Victoria|**2**
3818125752|3818125752|street|housenumber|Royal Victoria|**6**
3818126164|3818126164|street|housenumber|Royal Victoria|**8**
3818123785|3818123785|street|housenumber|Royal Victoria|**10**
3818123805|3818123805|street|housenumber|Royal Victoria|12
3818123826|3818123826|street|housenumber|Royal Victoria|**14**
3818123853|3818123853|street|housenumber|Royal Victoria|**16**
3818125669|3818125669|street|housenumber|Royal Victoria|**18**
3818125689|3818125689|street|housenumber|Royal Victoria|**20**
3818125706|3818125706|street|housenumber|Royal Victoria|**22**
3818125718|3818125718|street|housenumber|Royal Victoria|**24**
3818125723|3818125723|street|housenumber|Royal Victoria|**25**
3818125724|3818125724|street|housenumber|Royal Victoria|**26**
3818125725|3818125725|street|housenumber|Royal Victoria|**27**
3818125726|3818125726|street|housenumber|Royal Victoria|**28**
3818125727|3818125727|street|housenumber|Royal Victoria|**29**
3818125728|3818125728|street|housenumber|Royal Victoria|**30**
3818125729|3818125729|street|housenumber|Royal Victoria|**31**
3818125730|3818125730|street|housenumber|Royal Victoria|**32**
3818125731|3818125731|street|housenumber|Royal Victoria|**33**
3818125732|3818125732|street|housenumber|Royal Victoria|**34**
3818125733|3818125733|street|housenumber|Royal Victoria|**35**
3818125734|3818125734|street|housenumber|Royal Victoria|**36**
3818125735|3818125735|street|housenumber|Royal Victoria|**37**
3818125736|3818125736|street|housenumber|Royal Victoria|**38**
3818125737|3818125737|street|housenumber|Royal Victoria|**39**
3818125738|3818125738|street|housenumber|Royal Victoria|**40**

3818125739|3818125739|street|housenumber|Royal Victoria|**41** 3818125740|3818125740|street|housenumber|Royal Victoria|**42** 3818125741|3818125741|street|housenumber|Royal Victoria|**43** 3818125742|3818125742|street|housenumber|Royal Victoria|**44** 3818125743|3818125743|street|housenumber|Royal Victoria|**45** 3818125744|3818125744|street|housenumber|Royal Victoria|**46** 3818125745|3818125745|street|housenumber|Royal Victoria|**47** 3818125746|3818125746|street|housenumber|Royal Victoria|**48** 3818125747|3818125747|street|housenumber|Royal Victoria|**50** 3818125748|3818125748|street|housenumber|Royal Victoria|**52** 3818125749|3818125749|street|housenumber|Royal Victoria|**54** 3818125750|3818125750|street|housenumber|Royal Victoria|**56** 3818125751|3818125751|street|housenumber|Royal Victoria|**58** 3818125753|3818125753|street|housenumber|Royal Victoria|**60** 3818125754|3818125754|street|housenumber|Royal Victoria|**62** 3818125755|3818125755|street|housenumber|Royal Victoria|**64** **…**	

Note: that the first 11 homes are evenly numbered, **there is no #4**. Then, the homes are numbered sequentially thru 48, followed by a repetition of the even numbered pattern. This peculiarity is illustrated by the pictures of some mailboxes in my community. Thus the OSM data mirrors reality, so my confidence in the validity of the data, for this locality, is reinforced!

Furthermore, **Royal Victoria [street] turns into Saint James [street]** in our housing development. Again, the OSM data perfectly mirrors reality - as illustrated by juxtaposing results from the above SQL query with results from a similar query for Saint James [street].

3818125756|3818125756|street|housenumber|Royal Victoria|66
3818126157|3818126157|street|housenumber|Royal Victoria|68
3818126158|3818126158|street|housenumber|Royal Victoria|70
3818126159|3818126159|street|housenumber|Royal Victoria|72
3818126160|3818126160|street|housenumber|Royal Victoria|74
3818126161|**3818126161**|street|housenumber|**Royal Victoria**|**76**

3818126162|**3818126162**|street|housenumber|**Saint James**|**77**
3818126163|3818126163|street|housenumber|Saint James|79
3818126165|3818126165|street|housenumber|Saint James|81

I conclude that at least for my neighborhood the data appears to be accurate and complete!

Consistency and Uniformity

City names
Finding problems:

SQL Query	**Result**
SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) tags WHERE tags.key LIKE “city” GROUP BY tags.value ORDER BY count DESC;	Irvine|**24963** Tustin|**2419** Lake Forest|1360 Santa Ana|29 Newport Beach|21 Costa Mesa|1 **Tustin, CA|1** **irvine|1**

Fixing problems:

For this project, I've chosen to fix the *city name* anomalies programmatically in the Python code before loading the data into the SQL tables, as follows:

audit.py program code snippet	***data.py*** program code snippet
expected_cities = ["Irvine", "Santa Ana", "Newport Beach", "Tustin", "Lake Forest", "Costa Mesa", "Fountain Valley"] city_mapping = { "Tustin, CA": "Tustin", "irvine": "Irvine" }	

def **is_city_name**(elem):
  return (elem.attrib['k'] == "addr:city")
 	
def **update_city_name**(city_name, city_mapping):
  for key in city_mapping:
  if key in city_name:
  city_name = string.replace(city_name,key,city_mapping[key])
  return city_name
	# Cleaning and loading values of various keys
	
	elif **is_city_name**(secondary):
	  city_name = secondary.attrib['v']
	  city_name = **update_city_name**(city_name, city_mapping)
	  new['value'] = city_name
	  print city_name

Street Names and numbers

Finding problems:

SQL Query
select street_name.id, house_no.id, street_name.key, house_no.key, street_name.value, house_no.value from nodes_tags as street_name, nodes_tags as house_no where street_name.key = "street" and house_no.key = "housenumber" and street_name.id = house_no.id order by CAST(house_no.value as INTEGER) limit 8;

Result

1211580217|1211580217|street|housenumber|**E First St #900|Xerox Corporation, 1851**
3891135366|3891135366|street|housenumber|**Rampage Lane|A4032**
1211580209|1211580209|street|housenumber|Premier Place|1
3502017257|3502017257|street|housenumber|Azalea|1
3502017258|3502017258|street|housenumber|Photinia|1
3502030811|3502030811|street|housenumber|Brockton|1
3502060112|3502060112|street|housenumber|Iris|1
3502060113|3502060113|street|housenumber|New Dawn|1

Fixing *Street Name* Problems:

Again, for this project, I chose to fix the *street name* anomalies programmatically in the Python code, before loading the data into the SQL tables, as follows:

audit.py program code snippet	***data.py*** program code snippet

def **is_street_name**(elem):
  return (elem.attrib['k'] == "addr:street")
 	
def **update_name**(street_name, mapping):
  for key in mapping:
  if key in street_name:
  better_name = re.sub(r'#\d+',"",street_name)
  street_name = string.replace(better_name,key,
  mapping[key])
  return street_name
	# Cleaning and loading values of various keys
	
	if **is_street_name**(secondary):
	  street_name = secondary.attrib['v']
	  street_name = update_name(street_name,
	  mapping)
	  new['value'] = street_name
	  print street_name

Fixing *Street Number* Problems:

In this case I opted to use SQL to correct the street number for the local **Xerox Corp**. office. I verified the validity of address using Google, then corrected the street number to conform to the schema, as follows:

SQL Updates
UPDATE nodes_tags SET value = replace (value, "Xerox Corporation, 1851", "1851") WHERE value LIKE "Xerox Corporation, 1851";

Result
1211580217|1211580217|street|housenumber|**E First Street**|**1851**

In the case of **Rampage Lane**, I had to determine what should be the correct street number by inspecting neighboring street numbers as follows:

SQL Query
SELECT street_name.id, house_no.id, street_name.key, house_no.key, street_name.value, house_no.value
FROM nodes_tags as street_name, nodes_tags as house_no
WHERE street_name.key = "street"
and house_no.key = "housenumber"
and street_name.value = "Rampage Lane"
and street_name.id = house_no.id
ORDER BY CAST(house_no.value as INTEGER);

Result
3891135366|3891135366|street|housenumber|Rampage Lane|**A4032**
3891135357|3891135357|street|housenumber|Rampage Lane|4042
3891135375|3891135375|street|housenumber|Rampage Lane|4051
3891135352|3891135352|street|housenumber|Rampage Lane|4052
3891135344|3891135344|street|housenumber|Rampage Lane|4062

UPDATE nodes_tags SET value = replace (value, "A4032", "4032") WHERE value LIKE "A4032";

Finding postcode consistency problems:

SELECT tags.value, COUNT(*) as count FROM (SELECT * FROM nodes_tags UNION ALL	**…** **"CA 92614",2** 92619,1

SELECT * FROM ways_tags) tags WHERE tags.key='postcode' GROUP BY tags.value ORDER BY count DESC;	92626,1 92706,1 92708,1 92709,1 **92780-4629,1** **"CA 92603",1** **"CA 92612",1** **"CA 92618",1**

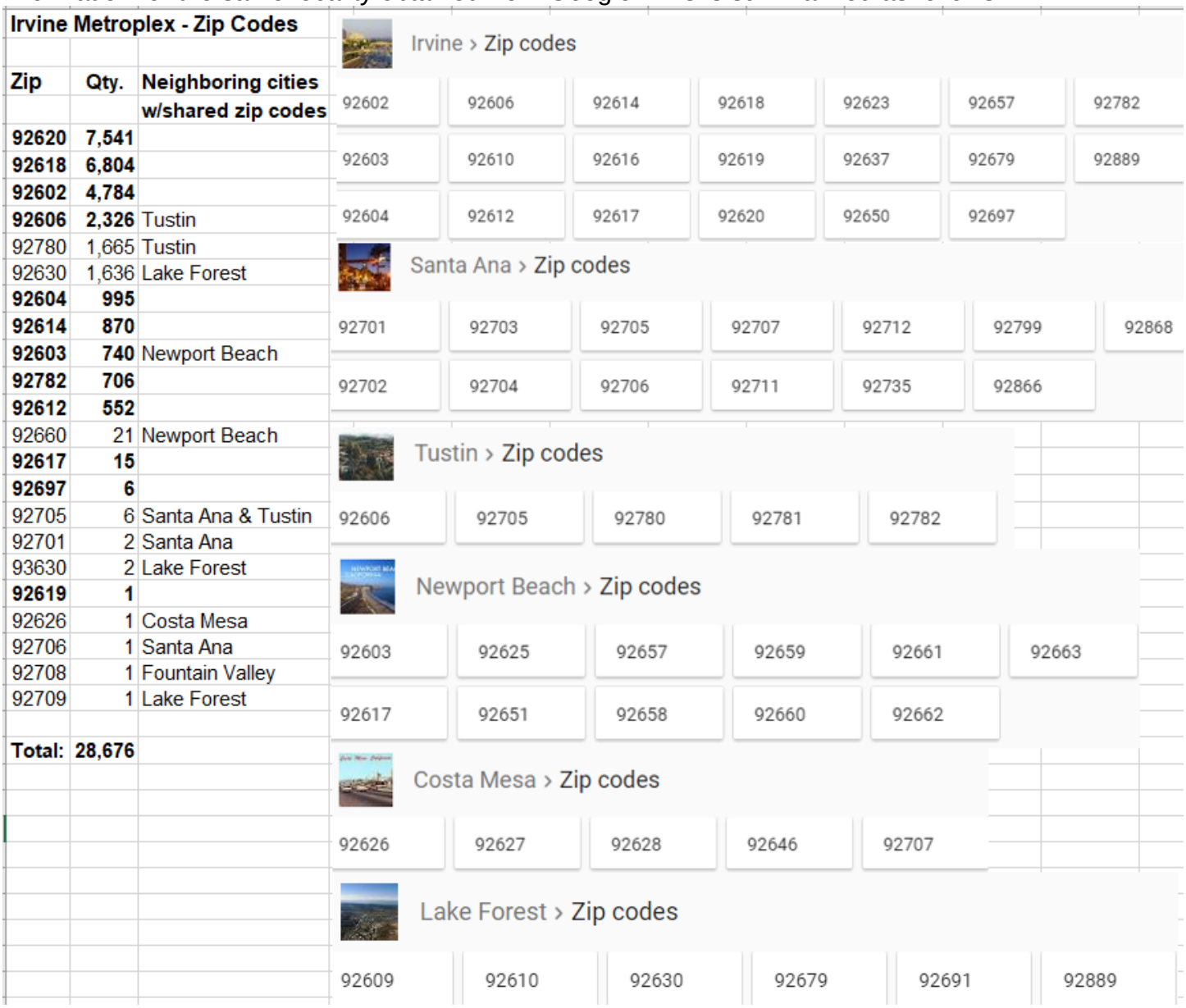
Fixing postcode consistency problems:

UPDATE nodes_tags SET value = replace(value, "CA 92618", "92618") WHERE value LIKE "CA 92618";
UPDATE ways_tags SET value = replace(value, "92780-4629", "92780") WHERE value LIKE "92780-4629";

I then queried the corrected OSM zip code data and performed an accuracy reconciliation with zip information for the same locality obtained from Google. This is summarized as follows:

Irvine Metroplex - Zip Codes

Zip	Qty.	Neighboring cities w/shared zip codes
92620	**7,541**	
92618	**6,804**	
92602	**4,784**	
92606	**2,326**	Tustin
92780	1,665	Tustin
92630	1,636	Lake Forest
92604	**995**	
92614	**870**	
92603	**740**	Newport Beach
92782	**706**	
92612	**552**	
92660	21	Newport Beach
92617	**15**	
92697	**6**	
92705	6	Santa Ana & Tustin
92701	2	Santa Ana
93630	2	Lake Forest
92619	**1**	
92626	1	Costa Mesa
92706	1	Santa Ana
92708	1	Fountain Valley
92709	1	Lake Forest
Total:	**28,676**	



Data Statistics
This section contains basic statistics about the Irvine metroplex dataset:

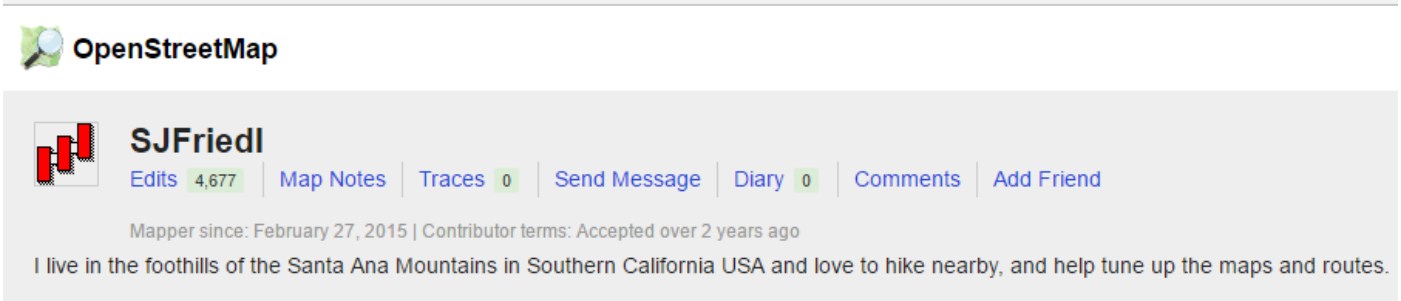
Irvine_OSM_full.txt ..... 55.5 MB
OSM_Project.db ..... 2.7 MB
nodes.csv ..... 18.4 MB
nodes_tags.csv ..... 4.5 MB
ways.csv ..... 2.0 MB
ways_tags.csv ..... 4.9 MB
ways_nodes.cv ..... 6.2 MB

Number of nodes SELECT COUNT(*) FROM nodes; **218462**	**Number of ways** SELECT COUNT(*) FROM ways; **34063**
Number of distinct users SELECT COUNT(DISTINCT(n_w.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) as n_w; **436**	**Number of users with only one post** SELECT COUNT(*) FROM (SELECT n_w.user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) as n_w GROUP BY n_w.user HAVING num=1) as unipost; **95**

Top 6 users with the most posts

SQL Query	**Result**
SELECT n_w.user, COUNT(*) as num FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) as n_w GROUP BY n_w.user ORDER BY num DESC LIMIT 6;	**SJFriedl|140090** **Aaron Lidman|22422** **ponzu|11136** **RichRico|5086** **The Temecula Mapper|4420** **karitotp|4333**

Observations/Suggestions
Clearly *SJFriedl* is a prolific user/contributor to the map. Who is this person, and what motivates him/her to contribute so much?
From the OSM web site I found the following user/contributor information:



For those with similar ‘mapper’ interests, it would be great to form an active user/contributor group, perhaps mentored by SJ Friedl, to continue to improve the quality of the OSM data for the Irvine metroplex area. OSM features such as *Map Notes*, *Diary*, *Comments*, and *Add Friend* makes it easy to contact and to collaborate with SJ Freidl and/or other mappers. Since, there is a fairly % of users

who have only made a single post it may be difficult so generate sufficient interest to sustain a user group

Suggestion for Improving the Quality of the Data

Couple the OSM data update process to ride share organizations', such as Uber's and/or Lyft's, navigation systems, or to an app that drivers could download to their personal smartphones. Each time an Uber/Lyft driver is called to a location the driver can verify that the OSM data is accurate, uniform, and complete. Points could be awarded to drivers and appear on their Uber/Lyft profiles. Civic minded philanthropic organizations, such as the Bloomberg Foundation, may be open to provide funding to award prizes for points achievements.

Top 10 amenities

SQL Query	Result
SELECT value, COUNT(*) as num FROM nodes_tags WHERE key='amenity' GROUP BY value ORDER BY num DESC LIMIT 10;	**restaurant|74** **bicycle_parking|53** **cafe|29** **bench|26** **drinking_water|26** **fast_food|23** **toilets|18** **fountain|12** **school|9** **bank|8**

Identify Names of a few Cafes

SQL Query	Result
SELECT distinct(amenity_type.id), amenity_type.value, amenity_name.value FROM nodes_tags as amenity_type, nodes_tags as amenity_name, nodes_tags as amenity_cat WHERE amenity_type.id = amenity_name.id and amenity_name.key = "name" and amenity_type.value = ”cafe” and amenity_cat.key = “amenity” GROUP BY amenity_type.id LIMIT 10;	**370225109|cafe|Starbucks** **417227407|cafe|Starbucks** **417227408|cafe|Cafe Expresso** **635507696|cafe|Tapioca Express** **635507701|cafe|Caf ├⌐ Brassiere** **994020246|cafe|Peet's Coffee & Tea** **1144103647|cafe|Starbucks** **1150448614|cafe|Peet's Coffee** **1151772767|cafe|Starbucks** **1153885190|cafe|Brueggers Bagels**

Top 10 Shops

SQL Query	Result

SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
  JOIN (SELECT DISTINCT(id) FROM nodes_tags) as type
  ON nodes_tags.id=type.id
WHERE nodes_tags.key=”shop”
GROUP BY nodes_tags.value
ORDER BY num DESC
	beauty|11
	convenience|5
	clothes|4
	hairdresser|4
	supermarket|4
	dry_cleaning|3
	sports|3
	bicycle|2

LIMIT 10;	**books|2** **florist|2**

Identify Names of Convenience Stores

SQL Query	Result
SELECT distinct(amenity_type.id), amenity_type.value, amenity_name.value FROM nodes_tags as amenity_type, nodes_tags as amenity_name, nodes_tags as amenity_cat WHERE amenity_type.id = amenity_name.id and amenity_name.key = "name" and amenity_type.value = ”convenience” and amenity_cat.key = “shop” GROUP by amenity_type.id	**754619957|convenience|Sand Canyon Service Station** **3134614835|convenience|Circle K** **3134653383|convenience|7-Eleven** **4821252721|convenience|711** **4881684557|convenience|7-Eleven**

Conclusion.

This project has introduced me to OSM. This is a very useful, community friendly, resource. Based on my queries I note that there remain data inconsistencies such as *7-Eleven* vs. *711*, or data errors such as *Caf ├⌐ Brassiere*. Now, I feel much better equipped to become an OSM user/contributor. I’ve also discovered that there is an easy way to meet up with other user/contributors via resources made available via the OSM web site.