Project 3: Wrangle OpenStreetMap (Richmond)

In this project, the OpenStreetMap of Richmond was explored. Overall, the data was relatively organized, however there were still several inconsistencies that were dealt with, such as finding and fixing problematic keys, updating street types, and standardizing phone numbers. After the data was cleaned and shaped into CSVs, it was imported to SQL database for querying. Various queries such as top amenities, zipcodes with most restaurants, most common coffee shops, and most common natural attributes revealed that Richmond is a relatively diverse standard American suburb.

PART 1: DATA WRANGLING

Wrangling Prep: Downsize File to Sample Size

The original data file was very big (131.7 MB) making it difficult to work with. Therefore it was downsized to a sample file (66.6 MB) by taking every other top element. This sample file was still pretty big, so I parsed the file even further taking every 100th element of the sample file and used that information to explore the data before going back to the sample file to make my official cleaning.

Wrangling Problem 1: Fixing Keys with a Space to an Underscore

A search through the keys of the elements revealed that most keys were in the correct spelling format, with most having only lower-case letters or a colon. However, a couple of the keys had problematic characters, which were updated to have an underscore instead of a space.

I used the code in the case study, combined with the one below to update the problem characters.

```
def fix_problemchars(osmfile): # replaces the space in the problemChar with underscores
    for _, element in ET.iterparse(osmfile):
        if element.tag == 'tag':
            k = element.attrib['k']
            probChars = problemchars.search(k)
            if probChars:
                print 'BEFORE'
                print k
                k = k.replace(probChars.group(),'_')
                print 'AFTER'
                print k
```

Wrangling Problem 2: Update Street Types

To make the street types more consistent, the abbreviated street names were updated to the full spelling (e.g., Ave to Avenue and Tnpk to Turnpike).

```
mapping = { "Ave": "Avenue",
            "Dr": "Drive",
            "Pkwy": "Parkway",
            "Rd": "Road",
            "St": "Street",
            "Tnpk": "Turnpike"
            }

def update_name(name, mapping):
    for key in mapping:
        if re.search(key, name):
            name = name.replace(key,mapping[key])

    return name
```

```
def update_street_types(osmfile):
    street_types = audit(osmfile)
    old_street_names = []
    updated_street_names = []
    for k in street_types.keys():
        for street in street_types[k]:
            old_street_names += [street]
            updated_street_names += [update_name(street, mapping)]
    return "old: ", old_street_names, "updated: ", updated_street_names
```

Wrangling Problem 3: Standardizing Phone Numbers

Certain locations, such as businesses and buildings, had phone numbers listed. However, the phone numbers came in various formats (e.g., +1 804 123 4567 and (804) 123-4567). To make the phone numbers more standardardized, special characters were removed and spaces separating numbers were changed to hyphens (e.g., 1-804-123-4567).

```
symbols_re = re.compile(r'[=\+\(\)/&<>;\'"\?%#$@\,\.\t\r\n]') # looks for special characters
space_re = re.compile(r'[ ]') # looks for space

# Update phone number to standardized form (eg. (804) 123-4567 to 804-123-4567)
def update_phone(number):
    updated_number = re.sub(symbols_re, '', number)
    updated_number = re.sub(space_re, '-', updated_number)
    return updated_number

# Read file, process update_phone, and print list to see the numbers that were changed
def process_phone(osmfile):
    osm_file = open(osmfile, "r") # open file and read
    num_before = []
    num_after = []
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node": # if the tag is node
            for tag in elem.iter("tag"): # for all tags with 'tag'
                if tag.attrib['k']=='phone': # if the key is 'phone'
                    num_before.append(tag.attrib['v']) # add to num_before to see prior to change
                    after = update_phone(tag.attrib['v']) # update phone number to standardized form
                    num_after.append(after) # see numbers after change
    osm_file.close()
    return 'Before: ', num_before, 'After: ', num_after
```

```
Out[157]: ('Before: ',
           ['804 276-3193',
            '(804) 864-2992',
            '(804) 643-8818',
            '(804) 644-2466',
            '(804) 644-5465',
            '(804) 643-8080',
            '804-648-3734',
            '(804) 343-3647',
            '(804) 367-2323',
            '804-323-2550',
            '804-796-4242',
            '804-323-2600',
            '+1 804-359-6688',
```

```
           'After: ',
           ['804-276-3193',
            '804-864-2992',
            '804-643-8818',
            '804-644-2466',
            '804-644-5465',
            '804-643-8080',
            '804-648-3734',
            '804-343-3647',
            '804-367-2323',
            '804-323-2550',
            '804-796-4242',
            '804-323-2600',
            '1-804-359-6688',
```

PART 2: QUERIES IN SQL

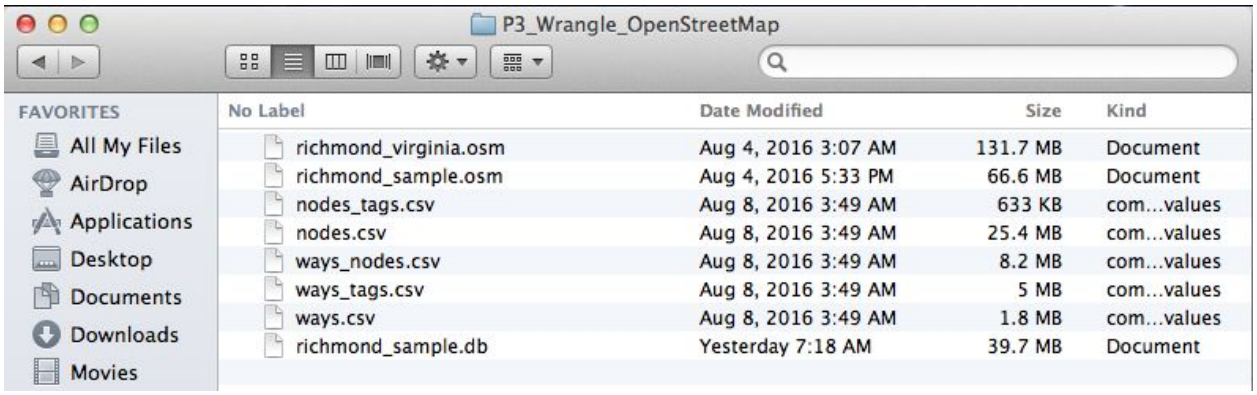
Query Prep: Import Cleaned CSV to SQL Database

After being cleaned and formatted as CSV files, the data was imported to the SQL database for querying.

Query 1: Overview

An overview query of the data showed that there were:

- 300,985 nodes
- 31,070 ways
- 351 unique users
- Top users were: woodpeck_fixbot (127,228 entries), RVA_101 (61,305 entries), and CynicalDooDad (33,881 entries)




```
# Find number of nodes_2: *I accidentally added the wrong info to nodes table, so use nodes_2
QUERY = "SELECT COUNT(*) FROM nodes_2;"
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(300985,)]
```

```
# Find number of ways:
QUERY = "SELECT COUNT(*) FROM ways;"
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(31070,)]
```

```
# Find number of unique users:
QUERY = "SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes_2 UNION ALL SELECT uid FROM ways) e;"
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(351,)]
```

```
# Find top 10 contributing users:
QUERY = '''
SELECT e.user, COUNT(*) as num
FROM (SELECT user FROM nodes_2 UNION ALL SELECT user FROM ways) e
GROUP BY e.user
ORDER BY num DESC
LIMIT 10;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'woodpeck_fixbot', 127228), (u'RVA_101', 61305), (u'CynicalDooDad', 33881), (u'Omnific', 25616), (u'gpstrails', 19
566), (u'42429', 10039), (u'TIGERcnl', 5882), (u'taber', 5861), (u'bot-mode', 5455), (u'daddyklee', 4669)]
```

Query 2: Top zipcodes and top cities in Richmond Area

The top city in the Richmond area, was unsurprisingly, Richmond. However, there were a few neighboring cities included, such as Glen Allen Midlothian. The top zipcode listed for Richmond was 23220, which is the area of the Virginia Commonwealth University (VCU) campus.

```
# Top Cities
QUERY = '''
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;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'Richmond', 135), (u'Glen Allen', 28), (u'Midlothian', 4), (u'Bon Air', 2), (u'Chesterfield', 2), (u'Sandston',
2), (u'1', 1), (u'3', 1), (u'Downingtown ', 1), (u'None', 1), (u'glen Allen', 1), (u'no', 1), (u'richmond', 1)]
```

```
# Top zipcodes
QUERY = '''
SELECT tags.value, COUNT(*) as count
FROM (SELECT * FROM nodes_tags
      UNION ALL
      SELECT * FROM ways_tags) tags
WHERE tags.key='postcode'
GROUP BY tags.value
ORDER BY count DESC;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'23220', 47), (u'23219', 32), (u'23059', 16), (u'23060', 12), (u'23230', 12), (u'23221', 10), (u'23223', 7), (u'23
235', 5), (u'23112', 2), (u'23114', 2), (u'23116', 2), (u'23150', 2), (u'23224', 2), (u'23236', 2), (u'23832', 2),
(u'19335', 1), (u'23221-3504', 1), (u'23222', 1), (u'23226', 1), (u'23227-1107', 1), (u'23233', 1), (u'23236-3103',
1), (u'23238', 1), (u'23284', 1), (u'23298', 1)]
```

Query 3: Exploring the Food and Coffee Scene

Food seems to be quite popular in Richmond as restaurant, fast food, and cafe all made it in the top ten amenities in Richmond. Richmond has a relatively diverse cuisine with American, Chinese, Italian, and Mexican on the list of most common cuisines. The zipcodes with the most restaurants were 23220, 23219, and 23230. Which is unsurprising since they are the zipcodes of the VCU campus, downtown area, and main street, respectively. The top cafes were the big chains of Starbucks, Dunkin' Donuts, and Tropical Smoothie Cafe.

```
# Top ten appearing amenities
QUERY = '''
SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='amenity'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'restaurant', 212), (u'place_of_worship', 154), (u'school', 119), (u'fast_food', 94), (u'fuel', 46), (u'bank', 3
9), (u'cafe', 32), (u'grave_yard', 32), (u'fire_station', 27), (u'pharmacy', 24)]
```

```
# Top ten cuisines
QUERY = '''
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='cuisine'
GROUP BY nodes_tags.value
ORDER BY num DESC;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'american', 8), (u'chinese', 7), (u'italian', 7), (u'mexican', 6), (u'sushi', 4), (u'pizza', 3), (u'regional', 3),
(u'thai', 3), (u'ice_cream', 2), (u'japanese', 2), (u'sandwich', 2), (u'asian', 1), (u'bbq', 1), (u'bistro', 1), (u'b
reakfast', 1), (u'burger', 1), (u'french', 1), (u'indian', 1), (u'korean', 1), (u'mediterranean', 1), (u'seafood',
1), (u'southern', 1), (u'tacos', 1)]
```

```
# Zipcodes with most restaurants
QUERY = '''
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='restaurant') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='postcode'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 5;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'23220', 17), (u'23219', 9), (u'23230', 3), (u'23221', 2), (u'23223', 2)]
```

```
# Top five cafes
QUERY = '''
SELECT nodes_tags.value, COUNT(*) as num
FROM nodes_tags
    JOIN (SELECT DISTINCT(id) FROM nodes_tags WHERE value='cafe') i
    ON nodes_tags.id=i.id
WHERE nodes_tags.key='name'
GROUP BY nodes_tags.value
ORDER BY num DESC
LIMIT 5;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'Starbucks', 10), (u'Dunkin Donuts', 2), (u'Tropical Smoothie Cafe', 2), (u'821 Cafe', 1), (u'Bell Cafe', 1)]
```

Query 4: Top Natural Attributes

The most common ‘natural’ attribute in Richmond were trees.

```
# Top ten 'natural' keys
QUERY = '''
SELECT value, COUNT(*) as num
FROM nodes_tags
WHERE key='natural'
GROUP BY value
ORDER BY num DESC
LIMIT 10;
'''
cur.execute(QUERY)
result = cur.fetchall()
print result
```

```
[(u'tree', 1722), (u'peak', 3), (u'cliff', 1), (u'wetland', 1)]
```

PART 3: CONCLUSIONS

To better explore the data and the area of Richmond, further queries could look into the zipcodes with the top amenities, such as places of worship or schools. A closer look of the area, could also explore the particular streets that have the most amenities. However, it should be taken into account that the data is not complete and many places are not listed at all or are not fully listed with all features, such as cuisine type and street name.

To solve this problem and improve the dataset, people could try using the GPS on their wearables such as Fitbit and Apple Watch to map data. By doing so, they could be mapping data everywhere they go and would increase the OpenStreetMap information and accuracy. Although this would generate a significant amount of data, there may be some privacy issues. By having their location recorded at all times and uploaded publicly into OpenStreetMap, people’s privacies could be compromised.