Wrangle OpenStreetMap Data: Rubric and Answers

By Callie Lamb

Code Functionality

Criteria - Does the code function as expected?

Specifications - Final project code functionality reflects the description in the project document.

> **Answer** - The code functionality meets the specifications for this project.

Code Readability

Criteria - Does the project utilize good coding practices?

Specifications - Final project code follows an intuitive, easy-to-follow logical structure.

> **Answer** - The code follows an intuitive, easy-to-follow structure.

Criteria - Is the code commented in a way that is useful and not superfluous?

Specifications - Final project code that is not intuitively readable is well-documented with comments.

> **Answer** – Yes, there are comments throughout the code explaining what the code is doing if it is not intuitive. Variable names and function names are named in an intuitive manner.

Problems Encountered in Your Map

Criteria - Does the project document the challenges encountered during the wrangling?

Specifications - Student response shows an understanding of the process of auditing, and ways to correct or standardize the data, including dealing with problems specific to the location, e.g., related to language or traditional ways of formatting.

> **Answer** – While exploring the data I found that way and node tags with key values of "addr:street", "name", tiger:name_type", and "tiger:name_type_1" all had similar street name endings and problems. Such as:
>
> - Some entries were all lowercase, some were all uppercase, and some had every first letter of each word capitalized.
> - Inconsistentnt labeling of street types (ie. Ave and Avenue for Avenue, Dr, Drive for Drive ect…)

- When I used SQL to find the top tourism types, I noticed that words are connected by an underscore (ie. picnic_site, camp_site). If I had more time, I'd like to investigate why this is and if this underscore is not needed, programmatically remove those, and capitalize the beginning of each word. I suspect other tags have this same problem as there are a lot of similarities between some tags.
- It would beneficial if when users are entering data, there is a standard data entry format or dropdowns to choose from. Or perhaps the data could be programmatically cleaned after it has been entered, but before it is written to the OSM database.

Criteria - Is data cleaned programmatically?

Specifications - Some of the problems encountered during data audit are cleaned programmatically.

Answer – Yes, the problems I encountered from the question above I was able to clean programmatically.

I made them more consistent by capitalizing each word and standardizing street acronyms to spell out the whole word. I was able to use the same function for all 4 of these tag keys and values. Below is from audit.py:

```
def is_street_name(elem):
    if elem.attrib['k'] == 'addr:street':
        update_name(elem)
    if elem.attrib['k'] == 'name':
        update_name(elem)
    if elem.attrib['k'] == 'tiger:name_type':
        update_name(elem)
    if elem.attrib['k'] == 'tiger:name_type_1':
        update_name(elem)

def update_name(tag):

    names = ['Adventures', 'Avenue', 'Basin', 'Cabin', 'Campsite', 'Circle', 'Court',
             'Creek', 'Drive', 'Lake', 'Lane', 'Lodge', 'Loop', 'Park', 'Parkway', 'Point',
             'Ranch', 'Ridge', 'River', 'Road', 'Stream', 'Street', 'Trail', 'Trailhead',
             'Volcano', 'Way']

    mapping = {'Ave': 'Avenue', 'Cir': 'Circle', 'Ct': 'Court', 'Dr': 'Drive',
               'Ln': 'Lane', 'Pky': 'Parkway', 'Rd': 'Road', 'Trl': 'Trail'}

    name = tag.attrib['v'].title()

    last_word = name.split()[-1]

    for key, value in mapping.items():
        if last_word == key:
            last_word = value
        if last_word != key:
            for word in names:
                if last_word == word:
                    pass
                if last_word != word:
                    pass
```

```
    new_name = name.replace(name.split()[-1], last_word)
    tag.set('v', new_name)

    return tag
```

Overview of the Data

Criteria - Is the OSM XML large enough?

Specifications - The OSM XML file is at least 50 MB uncompressed.

> **Answer** – The OSM XML "Yellowstone.osm" is 137 MB uncompressed which meets and exceeds the project specifications. The random "Sample.osm" that can be created is usually around 13 MB. I encountered issues during my exhaustive attempts to get an export of an OSM of that size. Every time I exported data from Overpass API, it would come in a compressed file format that I couldn't figure out how to deal with. Finally, in a last-ditch effort, I ran it through my create_sample.py, taking every "k-th" level element instead of every 10th level element. This did the trick. I wish this situation was simply documented in the OSM wiki.

Criteria - Are overview statistics of the dataset computed?

Specifications - Database queries are used to provide a statistical overview of the dataset, like: size of the file, number of unique users, number of nodes and ways, number of chosen type of nodes, like cafes, shops etc. Additional statistics not in the list above are computed. For SQL submissions some queries make use of more than one table.

> **Answer** – Yes, queries.py uses SQL queries to access the "Yellowstone_SQL_DB.db" (created in create_db.py). I have 9 queries that run and satisfy the stated specifications. The code and output are below:

```
import sqlite3


def number_of_nodes():
    result = cur.execute('SELECT COUNT(*) FROM nodes')
    return result.fetchone()[0]

def number_of_nodes_tags():
    result = cur.execute('SELECT COUNT(*) FROM nodes_tags')
    return result.fetchone()[0]

def number_of_ways():
    result = cur.execute('SELECT COUNT(*) FROM ways')
    return result.fetchone()[0]

def number_of_ways_tags():
    result = cur.execute('SELECT COUNT(*) FROM ways_tags')
    return result.fetchone()[0]

def unique_users_count():
    result = cur.execute('SELECT COUNT(distinct(uid)) FROM (SELECT uid FROM nodes UNION \
                          ALL SELECT uid FROM ways)')
    return result.fetchone()[0]
```

```
def top_contributing_users():
    result = cur.execute('SELECT e.user, COUNT(*) as num \
                          FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e \
                          GROUP BY e.user \
                          ORDER BY num DESC \
                          LIMIT 5')
    return result.fetchall()



def top_natural():
    result = cur.execute('SELECT e.value, COUNT(*) as num \
                          FROM (SELECT value FROM nodes_tags WHERE key = "natural" \
                                UNION ALL SELECT value FROM ways_tags WHERE \
                                key = "natural") e \
                          GROUP BY e.value \
                          ORDER BY num DESC \
                          LIMIT 5')
    return result.fetchall()

def top_tourism():
    result = cur.execute('SELECT e.value, COUNT(*) as num \
                          FROM (SELECT value FROM nodes_tags WHERE key = "tourism" \
                                UNION ALL SELECT value FROM ways_tags WHERE \
                                key = "tourism") e \
                          GROUP BY e.value \
                          ORDER BY num DESC \
                          LIMIT 5')
    return result.fetchall()

def top_path_type():
    result = cur.execute('SELECT e.value, COUNT(*) as num \
                          FROM (SELECT value FROM nodes_tags WHERE key = "highway" \
                                UNION ALL SELECT value FROM ways_tags WHERE \
                                key = "highway") e \
                          GROUP BY e.value \
                          ORDER BY num DESC \
                          LIMIT 5')
    return result.fetchall()


if __name__ == '__main__':

    con = sqlite3.connect("Yellowstone_SQL_DB.db")
    cur = con.cursor()
    print("\nNumber of Ways:\t\t\t", number_of_ways())
    print("Number of Nodes:\t\t", number_of_nodes())
    print("Number of Ways Tags:\t", number_of_ways_tags())
    print("Number of Nodes Tags:\t", number_of_nodes_tags())
    print("Number of Unique Users:\t", unique_users_count())
    print("\nTop Path Types:\t\t\t\t", top_path_type())
    print("Top Tourism Types:\t\t\t", top_tourism())
    print("Top Contributing Users:\t\t", top_contributing_users())
    print("Top Natural Geology Types:\t", top_natural())
```

Output (results are from using Yellowstone.osm):

```
Number of Ways:          23403
Number of Nodes:         729784
Number of Ways Tags:     38919
Number of Nodes Tags:    8680
Number of Unique Users:  316

Top Path Types:
[('service', 862), ('path', 610), ('footway', 383), ('track', 215),
('residential', 198)]

Top Tourism Types:
[('information', 230), ('viewpoint', 96), ('camp_site', 56), ('attraction',
56), ('picnic_site', 53)]

Top Contributing Users:
[('GeoDave5280', 479905), ('Gone', 49696), ('miraculixOSM', 41631),
('ottwiz', 26012), ('alexrudd (NHD)', 25202)]

Top Natural Geology Types:
[('wood', 10716), ('geyser', 697), ('water', 596), ('peak', 379),
('grassland', 307)]
```

Way to go GeoDave! I didn't think it was possible but with 479,905 contributions, perhaps GeoDave loves Yellowstone more than I do! I can see how this type of data could be a benefit to have on hand. For example, a state needs to plan a budget for fixing all picnic sites, they could query how many are in the park and get a rough idea of how much it will cost.

In addition to the SQL queries, I have a function in main.py that retrieves all project file names and sizes. It also calculates the entire size of the project. The function code and output for that are below:

```python
def file_sizes(path):
    project_path = lambda x: os.path.isfile(os.path.join(path, x))
    files_list = filter(project_path, os.listdir(path))

    # Creating a list of files and their sizes
    size_of_file = [(f, os.stat(os.path.join(path, f)).st_size) for f in files_list]

    num_list = []
    size_dict = {}

    for f, s in size_of_file:
        num_list.append(round(s / (1024 * 1024)))
        size_dict[f] = round(s / (1024 * 1024), 3)

    size_df = pd.DataFrame(list(size_dict.items()), columns=['FILE NAME', 'FILE SIZE (MB)'])
    size_df = size_df.to_string(index=False)
```

```
print("\nComplete Project Size:\t{} MB\n".format(sum(num_list)))
print(size_df)
```

Output (results are from using Yellowstone.osm):

```
Complete Project Size:  294 MB

            FILE NAME  FILE SIZE (MB)
             audit.py           0.001
         create_db.py           0.002
      create_sample.py          0.001
              main.py           0.008
             nodes.csv         60.439
        nodes_tags.csv          0.426
            queries.py          0.003
            Sample.osm         13.620
             schema.py          0.002
              ways.csv          1.408
         ways_nodes.csv        13.522
          ways_tags.csv         1.348
       Yellowstone.osm        137.975
Yellowstone_SQL_DB.db          66.250
```

Criteria - Are the database queries documented?

Specifications - The submission document includes the database queries and statistics from above.

Answer – Yes, the queries and results are documented in the screenshots above. They are also provided in my GitHub repository as “queries.py”.

Other Ideas About the Dataset

Criteria - Are ideas for additional improvements included?

Specifications - Submission document includes one or more additional suggestions for improving the data or its analysis. The suggestions are backed up by at least one investigative query.

Answer – Yes in a question/answer above I mentioned that when I used SQL to find the top tourism types, I noticed that words are connected by an underscore (ie. picnic_site, camp_site). If I had more time, I’d like to investigate why this is and if this underscore is not needed, programmatically remove those, and capitalize the beginning of each word. I suspect other tags have this same problem as there are a lot of similarities between some tags.

Criteria - Are benefits and problems with additional improvements discussed?

Specifications - Submission document includes thoughtful discussion about the benefits as well as some anticipated problems in implementing the improvement.

Answer – Yes, in an answer above I discuss a possible benefit of using this data. I also address some of the problems encountered and how I would programmatically fix them and how some of the known issues could be addressed during the data entry part from the users.

Thoroughness and Succinctness of Submission

Criteria - Is the submission long enough to answer the questions?

Specifications - Submission document is long enough to thoroughly answer the questions asked without giving unnecessary detail. A good general guideline is that the question responses should take about 3-6 pages.

Answer – Yes, the submission document length meets the stated specifications.