OpenStreet Map Data Case Study

Map Area

I have chosen to use the Melbourne, FL and the surrounding area as my subject for the map because this is where I live. I really love this area as it has beautiful beaches and for the most part very mild temperatures - although I will say you have to deal with humidity most of the time but especially the Summer time. This area is also known as the Space Coast because Cape Canaveral and Kennedy Space Center are a part of it.

Map Link: https://www.openstreetmap.org/export#map=11/28.2025/-80.6335 (https://www.openstreetmap.org/export#map=11/28.2025/-80.6335)

Data Overview

File sizes

Here are the details for the file sizes that were used in this project.

```
space_coast.osm ....... 54.5 MB
space_coast_tiny.osm... 5.6 MB
space_coast.db ........ 39 MB
nodes.csv ............. 19.5 MB
nodes_tags.csv ........ 713 KB
ways.csv .............. 1.9 MB
ways_tags.csv ......... 4.5 MB
ways_nodes.csv ........ 7 MB
```

The OSM XML file is greater than 50 MB uncompressed which meets the criteria for the project. While selecting the dataset from the map I purposely lassoed a portion of the map that would be close but over 50MB, because I just have a standard laptop (that has surprisingly worked well up to now), and just wanted to ensure I encountered no problem running the code to shape the CSV files. As it was the run time was around 15-20 minutes. I did make a 'tiny' version of the OSM XML file I used to test this code which only took a couple of minutes to run.

Tag Count

So I'll now start parsing through the OSM data to count the unique tags in the file.

```
import xml.etree.cElementTree as ET
import pprint

def count_tags(filename):
    tags = {}
    for event, elem in ET.iterparse(filename):
        if elem.tag in tags:
            tags[elem.tag] += 1
        else:
            tags[elem.tag] = 1
    return tags

tags = count_tags('space_coast.osm')
pprint.pprint(tags)
```

The results of the print statement:

```
{'bounds': 1,
 'member': 19339,
 'meta': 1,
 'nd': 290728,
 'node': 237933,
 'note': 1,
 'osm': 1,
 'relation': 348,
 'tag': 149860,
 'way': 31289}
```

For this project we focus on the 'node', 'way', and 'tag' tags. We will use this numbers to compare to database file that will be generated by the data.py code.

Data Cleaning and Building SQL Database

Look for Cleaning Opportunities

Before I process the data and add it into my database, I'm going to check the "k" value for each "tag" and see if there are any potential problems. Three regular expressions have been provided to identify the number of tags that match each regular expression.

The function 'key_type' within the script below will provide a count of each of four tag categories in a dictionary:

1) "lower" : for tags that contain only lowercase letters and are valid

2) "lower_colon" : for otherwise valid tags with a colon in their names

3) "problemchars" : for tags with problematic characters

4) "other" : for other tags that do not fall into the other three categories.

The result of check_tags.py is:

```
{'lower': 80708, 'lower_colon': 65150, 'other': 4002, 'problemchars': 0}
```

There are no problemchars in the tags of this dataset. 97% of the tags fall into either the 'lower' or 'lower_colon' category.

Deciding what to clean

I investigated a couple of key values to determine the cleanliness of them. First the addr:street types were collected with street_type_audit.py then the addr:postcode types were collected with postcodes_audit.py. There was a variety of issues with the street types where street where abbreviations were used a few times, the basic abbreviations like Avenue and Ave. etc. I also noticed there were street types that were valid however not in the expected list. For example Causeway is a common street type for this geography as there are many bridges and jetties that cross the Indian River lagoon that separate the bearer islands from the mainland. The postcodes were fairly clean. I researched the postcodes for Brevard County and all codes were valid for this area. I did notice on postcode that include 'FL', but it was the only one some I decided not to clean the postcodes.

Example of Street Types that needed to be cleaned:

```
'Ave': set(['Burdock Ave',
            'N Atlantic Ave',
            'North Atlantic Ave',
            'Pineapple Ave',
            'Town Center Ave',
            'W New Haven Ave']),
 'Ave.': set(['West New Haven Ave.']),
 'Ln': set(['Bogey Ln',
           'Breeze Ln',
           'Fire Fall Ln',
           'Hampton Park Ln',
           'Mollie Ln',
           'Patty Ln',
           'Rhonda Ln',
           'Sand Trap Ln',
           'Sanddune Ln',
           'Steeplechase Ln']),
 'Way': set(['Augusta Way',
             'Autumn Way',
             'Briarwood Way',
             'Charlie Corbeil Way',
             'Good Seasons Way',
             'Judge Fran Jamieson Way',
```

Postcodes:

```
{'32901': set(['32901']),
 '32903': set(['32903']),
 '32904': set(['32904', 'FL 32904']),
 '32905': set(['32905']),
 '32907': set(['32907']),
 '32909': set(['32909']),
 '32910': set(['32910']),
 '32920': set(['32920']),
 '32922': set(['32922']),
 '32931': set(['32931']),
 '32934': set(['32934']),
 '32935': set(['32935']),
 '32937': set(['32937']),
 '32940': set(['32940']),
 '32950': set(['32950']),
 '32951': set(['32951']),
```

Auditing and Preparing the Cleaning Code

To clean the street types I prepared the mapping list and update_name function within the street_type_pre_clean.py code. This two parts of the code will be included within the data.py code to perform the actual cleaning function. The cleaning is done before the CSV files are written.

The data.py code is used to generate the CSV files.

The CSV files are written using the schema provided by the schema.py code.

I encounter a bit of trouble getting the code to recognize cerberus. This was resolved by explicitly defining the appropriate path to the cerberus module. Once my working directory was defined the code was able to run.

Also during this step I failed to recognize the cleaning code needed to be added to the data.py code. My first run did not include the mapping list and update_name therefore no cleaning was done. It was discovered while doing some basic SQL queries founding lots of abbreviated street types.

Building the SQL Database

The build_database.py code was used to read in the CSV files and create space_coast.db file. This file will be used to perform SQL queries on the dataset.

SQL Queries - Statistical Overview of the Dataset

The following queries were performed on the space_coast.db.

Number of nodes

```
QUERY='''SELECT count(*)
        FROM nodes;'''
```

237933

Number of ways

```
QUERY='''SELECT count(*)
        FROM ways;'''
```

31289

Number of unique users

```
QUERY='''SELECT COUNT(DISTINCT(e.uid))
        FROM (select uid from nodes UNION ALL select uid from ways) as e'''
```

309

The number of nodes and ways tags matched the numbers from the check_tags.py code giving me a good feeling about the validity of the data for the analysis.

Top 10 contributing users

```
QUERY='''SELECT subq.user, COUNT(*) as num
        FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) subq
        GROUP BY subq.user
        ORDER BY num DESC
        LIMIT 10'''
```

```
Panther37..... 62575
ingalls..... 44219
LeTopographeFou... 24742
grouper..... 15826
MichaelG68..... 14477
Easky30..... 11128
AeroMatter..... 9597
NE2..... 8964
Perfal..... 6660
realjd..... 6463
```

Panther37 was the most prolific contributor for this data set. The name Panther could have come from the mascot of the local university Florida Institute of Technology (FIT).

Number of users appearing only once (having 1 post)

```
QUERY='''SELECT COUNT(*)
        FROM (SELECT e.user, COUNT(*) as num
        FROM (SELECT user FROM nodes UNION ALL SELECT user FROM ways) e
        GROUP BY e.user
        HAVING num = 1)'''
```

69

Top 10 Amenities

```
QUERY='''SELECT value, COUNT(*) as num
            FROM nodes_tags
            WHERE key="amenity"
            GROUP BY value
            ORDER BY num DESC
            LIMIT 10'''
```

```
restaurant..... 157
place_of_worship.... 133
school..... 109
fast_food..... 59
fuel..... 43
fire_station..... 40
fountain..... 40
cafe..... 36
police..... 21
bar..... 20
```

Restaurants, fast_food, bar, and cafe could be combined into one amenity perhaps called eating establishments which dominate this area of Florida. Florida is a destination location for tourist and it makes sense the service industry would be so dominant.

Additional Ideas for Improved Analysis

Some other ways I would improve my analysis is to go back and look at more complete list of keys to investigate the top 10 for the most efficient cleaning opportunities.

The following query provides the top ten list of the keys.

```
QUERY='''SELECT subq.key,  COUNT(*) as num
        FROM (SELECT key FROM nodes_tags UNION ALL SELECT key FROM ways_tags) subq
        GROUP BY subq.key
        ORDER BY num DESC
        LIMIT 10'''
```

Resulting in the following list:

```
0                highway  21058
1                   name  12027
2                 county   8228
3                   cfcc   8216
4              name_base   7929
5              name_type   7818
6               building   7759
7               reviewed   7210
8               zip_left   6739
9              zip_right   6381
```

For example I would start with the key=highway as the most target rich key to clean, then continue down the list.

The following query provides the top ten list of the values in the highway key.

```
QUERY='''SELECT subq.value,  COUNT(*) as num
        FROM (SELECT * FROM nodes_tags UNION ALL SELECT * FROM ways_tags) subq
        WHERE subq.key == 'highway'
        GROUP BY subq.value
        ORDER BY num DESC
        LIMIT 50'''
```

Resulting in the following list:

```
0         residential  8080
1             service  5345
2             footway  1914
3      turning_circle  1160
4     traffic_signals   669
5            bus_stop   568
6            crossing   520
7            tertiary   440
8             primary   410
9           secondary   346
```

Other Ideas about the Dataset

While reviewing the dataset I noticed several attributes that had a 'tiger' reference. Researching online and within the Udacity Forums I discovered that Tiger is an acroymn that stands for "Topologically Integrated Geographic Encoding and Referencing" system. It appears that attributes with the tiger notation were a duplicate within the same tags. The data that I cleaned, which was to make the street types consistent by using the full name instead of an abbreviation of some sort, was consistent amoung the duplicate Tiger data. The Tiger street type data was consistent as an abbreviation. So it would seem to me that based on this having a mapping system that referenced the Tiger data would yield data that required less cleaning. One would think Tiger data is a more reliable source as it's from the US Census Bureau so relying on it would not be a bad thing. However if you are analyzing area outside of the US when the Tiger data is not there - then it's back to the method that is not necessarily uniform (by using various other sources and users).

Also when I collected the street map data from the openstreetmaps link I really wanted to lasso the much of the barrier island in the space coast area. However I was bound by a rigid rectangle that didn't allow the flexibility to select with more precision the area that I wanted to analyze. So that would be an improvement that I would note - namely to allow a more free form lasso to enable the selection of particular areas. I guess there would be certain challenges to overcome in the implementation of this collection feature - a rectangle is very structured and deviating for this could be problematic if the same users were investigating/cleaning a common area of the map.

References

I used the following links in this project:

https://www.openstreetmap.org/export#map=11/28.2025/-80.6335 (https://www.openstreetmap.org/export#map=11/28.2025/-80.6335)

https://discussions.udacity.com/t/final-project-importing-cerberus-and-schema/177231?u=ct3963 (https://discussions.udacity.com/t/final-project-importing-cerberus-and-schema/177231?u=ct3963)

https://discussions.udacity.com/t/creating-db-file-from-csv-files-with-non-ascii-unicode-characters/174958/7 (https://discussions.udacity.com/t/creating-db-file-from-csv-files-with-non-ascii-unicode-characters/174958/7)

https://github.com/pratyush19/Udacity-Data-Analyst-Nanodegree/blob/master/P3-OpenStreetMap-Wrangling-with-SQL/query.py (https://github.com/pratyush19/Udacity-Data-Analyst-Nanodegree/blob/master/P3-OpenStreetMap-Wrangling-with-SQL/query.py)

https://github.com/lyvinhhung/Udacity-Data-Analyst-Nanodegree/blob/master/p3%20-%20Wrangle%20OpenStreetMap%20Data/P3%20-%20Data%20Wrangling%20with%20MongoDB.ipynb (https://github.com/lyvinhhung/Udacity-Data-Analyst-Nanodegree/blob/master/p3%20-%20Wrangle%20OpenStreetMap%20Data/P3%20-%20Data%20Wrangling%20with%20MongoDB.ipynb)

https://discussions.udacity.com/t/p3-final-project-changes-in-the-open-street-map/215578/8 (https://discussions.udacity.com/t/p3-final-project-changes-in-the-open-street-map/215578/8)

https://discussions.udacity.com/t/tiger-data-remove-or-keep/227554 (https://discussions.udacity.com/t/tiger-data-remove-or-keep/227554)

https://www.google.com/#q=postcode+for+brevard+county (https://www.google.com/#q=postcode+for+brevard+county)