OpenStreetMap Case Study

Introduction

My task is to investigate a data set from a location of my choosing from openstreetmap. I need identify problems, clean it and store the data in SQL. Then, I am to explore the data programmatically and propose ideas on how to improve the data set. This investigation is a practice project needed to complete the Data Analyst Nanodegree from Udacity.

Getting Started

This case has been documented using jupyter notebook DAND-Wrang-openStreetMap.ipynb. For review ing purposes and readability, I have organized the case by separating the codes, explaining the case on pdf and provided a sample data. Below are the files you need to follow this case:

- README.pdf: Contains my answ ers to the rubric and documents my data w rangling process.
- audit.py: Script that audits streetnames.
- data.py: Script that converts and cleans my data.
- auckland_new -zealand-sample.osm: Contains small part of the map region I used.

Location

Auckland MAP.
I chose Auckland, New Zealand as my location for my investigation because I have been planning to take a trip here for sometime now. I w ould like to take the opportunity to get myself familiar w ith the place by using it as an example for this project.

Problems Encountered in Your Map

To find and fix the streetnames in the map data, I ran the entire map data through a function that groups all street addresses into a dictionary according to the different variations used in the map. I filter the street names that I expect to be used and review the street addresses that I don't expect. Below are the problems that I w ill focus on for my audit.

- **Misspelled Names.** Some street names are spelled incorrectly like *Strreet*.
- **Incorrect Capitalization.** Some street names are not capitalized consistently like *road*.
- **Abbreviated Names.** Some of the street names are abbreviated. I w ould prefer not to use abbreviations of their names. Instead of *Hwy*, use *Highway*.
- **Problematic Format.** Street names w ith problematic characters w ill be ignored.

I start fixing the data set by using the function *update_name*, w hich revises the streetnames according to my specifications that are outlined in *mapping*. Once I am satisfied w ith my data, I process the data in an xml structure according to the example schema. I then turn xml into csv. Once the CSVs are generated and validated, I then import the data into an SQL database to begin my exploration.

Update Name

```
# This is what I use for reference when I update the street names.
mapping = {
  "street": "Street", "st": "Street", "st.": "Street", "rd": "Road",
  "road": "Road", "strret": "Street", "cr": "Crescent", "cresent": "Crescent",
  "crest": "Crescent", "hwy": "Highway", "ave": "Avenue", "plc,": "Place",
  "beach": "Beach", "way": "Way", s"ln": "Lane"
}

# Updates street name according to mapping dict.
def update_name(name, mapping):
  name_a = name.split(" ")

  for w in range(len(name_a)):
    if name_a[w].lower() in mapping.keys():
        name_a[w] = mapping[name_a[w].lower()]
  name = " ".join(name_a)

  return name
```

Shape Element

```
# Creates the XML structure. Updates street names.
def shape_element(element, node_attr_fields=NODE_FIELDS,
                  way_attr_fields=WAY_FIELDS,
                  problem_chars=PROBLEMCHARS,
                  default_tag_type='regular'):
    node_attribs = {}
    way_attribs = {}
    way_nodes = []
    tags = []

    if element.tag == 'node': # Nodes.
        for elem in NODE_FIELDS:
            if element.get(elem):
                node_attribs[elem] = element.attrib[elem]
            else: # Some nodes does not have attributes. This ignores them.
                return

        for elem in element:
            item = {}
            if PROBLEMCHARS.match(elem.attrib['k']): # Ignores problematic characters.
                continue
            elif LOWER_COLON.match(elem.attrib['k']): # If the element has a ':'.
                item['id'] = element.attrib['id']
                item['key'] = elem.attrib['k'].split(':')[1]
                item['type'] = elem.attrib['k'].split(':')[0]
                if is_street_name(elem):
                    item['value'] = update_name(elem.attrib['v'], mapping) # Updates street names.
                else:
                    item['value'] = elem.attrib['v']
            else: # For everythin else.
                item['id'] = element.attrib['id']
                item['key'] = elem.attrib['k']
                item['type'] = 'regular'
                if is_street_name(elem):
                    item['value'] = update_name(elem.attrib['v'], mapping)
                else:
                    item['value'] = elem.attrib['v']
            tags.append(item)

        return {'node': node_attribs, 'node_tags': tags}

    if element.tag == 'way': # Ways.
        i = 0 # counter for way_nodes elements. Since we do know how many they are.
        for elem in element.attrib:
            if elem in WAY_FIELDS:
```

```
                way_attribs[elem] = element.attrib[elem]

        for elem in element:
            item = {}
            item_nd = {}
            if elem.tag == "tag":
                if LOWER_COLON.match(elem.attrib["k"]):
                    item["id"] = element.attrib["id"]
                    item["key"] = elem.attrib["k"].split(":", 1)[1]
                    item["type"] = elem.attrib["k"].split(":", 1)[0]
                    if is_street_name(elem):
                        item['value'] = update_name(elem.attrib['v'], mapping)
                    else:
                        item["value"] = elem.attrib["v"]
                else:
                    item["id"] = element.attrib["id"]
                    item["key"] = elem.attrib["k"]
                    item["type"] = "regular"
                    if is_street_name(elem):
                        item['value'] = update_name(elem.attrib['v'], mapping)
                    else:
                        item["value"] = elem.attrib["v"]
                tags.append(item)

            if elem.tag == "nd":
                item_nd["id"] = int(element.attrib["id"])
                item_nd["node_id"] = int(elem.attrib["ref"])
                item_nd["position"] = i
                i += 1
                way_nodes.append(item_nd)

        return {"way": way_attribs, "way_nodes": way_nodes, "way_tags": tags}
```

SQL Queries

```
cur.execute('PRAGMA PAGE_SIZE;') # Programmatically checks the file size of the database.
page_size = cur.fetchone()
cur.execute('PRAGMA PAGE_COUNT;')
page_count = cur.fetchone()
database_size = page_size[0] * page_count[0]

cur.execute('SELECT COUNT(*) FROM ways;') # Counts the number of ways.
count_ways = cur.fetchall()

cur.execute('SELECT COUNT(*) FROM nodes;') # Counts the number of nodes.
count_nodes = cur.fetchall()

cur.execute('SELECT COUNT(DISTINCT(e.uid)) FROM (SELECT uid FROM nodes UNION ALL SELECT uid FROM ways) e;')
unique_users = cur.fetchall() # Counts the number of unique users.

cur.execute('SELECT COUNT(*) FROM nodes_tags WHERE key="tourism" and value="hotel"')
count_hotel = cur.fetchall() # Counts the number of hotels in the area.

cur.execute('SELECT COUNT(*) FROM nodes_tags WHERE key="tourism" and value="attraction"')
count_attraction = cur.fetchall() # Counts the number of attractions.

cur.execute('SELECT COUNT(*) FROM nodes_tags WHERE key="tourism" and value="museum"')
count_museum = cur.fetchall() # Counts the number of museums.

cur.execute('SELECT COUNT(*) \
            FROM \
                (SELECT value, COUNT(*) as num FROM nodes_tags WHERE key="tourism" \
                GROUP BY value ORDER BY num DESC LIMIT 100) u;')
tourism = cur.fetchall()

cur.execute('SELECT value, COUNT(*) as num FROM nodes_tags \
            WHERE key="tourism" \
            GROUP BY value \
            ORDER BY num DESC')
tourism_types = cur.fetchall()
```

```
print("Database Size: {}".format(database_size)) # Prints all of the SQL queries above.
print("Ways Count: {}".format(count_ways[0][0]))
print("Nodes Count: {}".format(count_nodes[0][0]))
print("Unique Users Count: {}".format(unique_users[0][0]))
print("Total Hotels: {}".format(count_hotel[0][0]))
print("Total Attractions: {}".format(count_attraction[0][0]))
print("Total Museums: {}".format(count_museum[0][0]))
print('Types of Tourism markers: {}'.format(tourism[0][0]))
pprint(tourism_types)
```

Overview of the Data

File Size

- auckland_new-zealand.osm: 658 MB
- project.db: 409 MB
- node_tags.csv: 3.67 MB
- ways.csv: 20 MB
- ways_nodes.csv: 84 MB
- ways_tags.csv: 86 MB
- nodes.csv: 246 MB

Counts

- Number of Unique Users: 953
- Number of Nodes: 2,913,110
- Number of Ways: 328,159
- Number of Hotels: 61
- Number of Attractions: 55
- Number of Museums: 13

Other Ideas about the Dataset

More Descriptive Attractions Markers for Tourists

It would be a lot more helpful if tourists attractions are given more detail on their labels. Currently, the tags are labeled as **attractions** are not as helpful as much for traveler to plan their trip. If the attractions have more detail such as **beach**, **monuments** or **nature**, then these would be more helpful for would-be travelers planning a visit to Auckland.

Currently, there are 18 types of tourism markers. One of the tourism sites are labeled as *attractions* with 55 markers on Auckland, New Zealand. If these markers are provided with more detail, it could be helpful information to travelers.

I suggest that if when users edit, there should be a snippet of text within text field, where the users would input the data, that would encourage the user to add more detail to their markers or labels. Instead of an empty field, it could instead say, "What can I see here?".

One of the problems that could result to this new feature, users might find the extra step as a burden. And this might deter users from contributing. In addition, the questions asked might be misinterpreted by users, which could lead

to inconsistency and subjective answers. Another problem that could result from this implementation is the variation of the input. With more detailed answers, it could open doors to more different interpretations. Some might use **monuments**, while others would use **statues**. This could result to increase variability and would make the data less usable.

Conclusions

Auckland, New Zealand map is well populated by contributions from active users. But because of the lack of guidelines or motivation to properly fill in the data is affecting the extent of details that each of the users is willing to input. If at the moment of data entry, there would be a descriptive text in a form of a question, users would be more obliged or encourage to place more detail on the data.

References

Carlward Source.
I have reviewed and patterned my analysis from carlward's sample_project.md that was recommended by the Udacity Project Description of this Submission.