DATA WRANGLING PROCESS:

Obtainment of raw openstreetmap dataset

1. Go to https://mapzen.com/data/metro-extracts/
2. Type in "Palo Alto, CA, USA"
3. Click "GET EXTRACT" button
4. From here, download the raw openstreetmap dataset in the OSM XML format, which is 16 MB

Auditing process

First, I should mention that I initially used the provided code to create a small sample of the larger dataset for convenience.

The nodes_tags and ways_tags lists are more likely than nodes, ways, and ways_nodes to contain various representations of data that are most in need of cleaning (since they contain qualitative information like keys, values, and types rather than quantitative information like latitude, longitude, user, etc.). Therefore, I intended to output several csv files containing every key, value, and type listed in nodes_tags and ways_tags lists. In addition, these csv files would list the number of instances of each key/value/type and order them from least to greatest. In total, there were 6 csv files to output:
(nodes_tags vs. ways_tags) x (key vs. value vs. type) = nodes_keys_audit.csv, nodes_values_audit.csv, nodes_types_audit.csv, ways_keys_audit.csv, ways_values_audit.csv, ways_types_audit.csv

After examining these various audit csv files, I came across a problem with the values of tags with 'phone' as there key. Namely, the phone numbers (i.e. the tag values) were irregular and had no consistent format (e.g. '+1 (650) 564-1024', '650-564-1024', '650.564.1024', etc.). I will cover how I solved this in the "cleaning process" section below.

Furthermore, I noticed that running the program with validate=True provides an error message pointing out that there is at least one instance where the "uid" and "user" fields of the nodes list are missing. We will have to deal with this in the cleaning process.

Cleaning process

As mentioned above, we need to find a way of dealing with data where the "uid" and "user" fields of the nodes list are missing. In order to accomplish this, I implemented the check_for_unfilled() function, which (1) checks if a field is missing, (2) checks if the missing field is either "uid" or "user", and then (3) replaces the missing field with the appropriate value. In the case of a missing "uid" field, I decided to have the missing value replaced with -1 (negative one), as all other "uid" values are positive integers; this allows us to uniquely mark such data for later identification. In the case of a missing "user" field, I decided to have the missing value replaced with "" (an empty string), as all other "user" values are non-empty strings (since Openstreetmap requires usernames be non-empty strings); this allows us to uniquely mark such data for later identification.

Next, I realized that there could be multiple variations for one particular ending of street names, as witnessed in the homework for this section of the program (e.g. “Ave.”, “Avenue”, “Ave”). Therefore, I implemented the clean_street_names() function to solve this possible problem.

Then there was the problem of phone numbers being irregular in format. I tackled this problem by utilizing regex to detect a wide range of conceivable phone number formats and then stripping the string concerned of all non-numeric characters (i.e. digits). Lastly, I checked if the length of the stripped string was more than 10 characters, in which case only the last 10 characters of the string were returned as a final result to replace the value of the tag.

However, I was faced a problem in dealing with some tags with ‘phone’ as their key that contained a non-phone number (e.g. wrongly input phone number, website address, etc.) as their value. See the screenshot below for some examples:

```
(DAND) Ians-MacBook-Pro-3:data ianscottknight$ python data.py
+1 408 650 965 7084
650368384
+1-650-938-NYNY
http://pacificcatch.com/locations/mountain-view/
http://www.prodigypress.com/
http://www.vtechmanufacturing.com/
http://usdls.com/
http://paulmartinsamericangrill.com/locations/mountain-view-ca-restaurant/
https://locations.laseraway.com/ca/mountainview/laser-hair-removal-mountainview-ca-15.html
```

Rather than simply let these values be erroneously labeled with the key ‘phone’, I decided to separate them into a new key of their own: ‘phone_irregular’. In this way, the proper phone numbers are cleaned and preserved while other non-phone numbers are removed but still preserved elsewhere.

DATA ANALYSIS PROCESS:

In order to properly analyze the data, I transferred the written csv files to an SQL database according to the provided schema found here: https://gist.github.com/swwelch/f1144229848b407e0a5d13fcb7fbbd6f

From this point, I set out to discover some overview statistics that might provide some interesting aspects of the area I decided to investigate. See below for a list of the several aspects I decided to investigate:

- Number of distinct users contributing to nodes and ways

```
sqlite> SELECT COUNT(*)
   ...> FROM (
   ...> SELECT DISTINCT *
   ...> FROM (
   ...> SELECT nodes.uid, ways.uid
   ...> FROM nodes
   ...> LEFT JOIN ways USING(uid)
   ...> UNION ALL
   ...> SELECT nodes.uid, ways.uid
   ...> FROM ways
   ...> LEFT JOIN nodes USING(uid)
   ...> WHERE nodes.uid IS NULL
   ...> )
   ...> );

952
sqlite>
```

- Number of distinct nodes

```
sqlite> SELECT COUNT(DISTINCT id)
   ...> FROM nodes;
1216571
sqlite>
```

- Number of distinct ways

```
sqlite> SELECT COUNT(DISTINCT id)
   ...> FROM ways;
135076
sqlite>
```

- Number of distinct restaurants

```
sqlite> SELECT COUNT(*)
   ...> FROM (
   ...> SELECT DISTINCT *
   ...> FROM (
   ...> SELECT nodes_tags.id, ways_tags.id, nodes_tags.value, ways_tags.value
   ...> FROM nodes_tags
   ...> LEFT JOIN ways_tags USING(id)
   ...> UNION ALL
   ...> SELECT nodes_tags.id, ways_tags.id, nodes_tags.value, ways_tags.value
   ...> FROM ways_tags
   ...> LEFT JOIN nodes_tags USING(id)
   ...> WHERE nodes_tags.id IS NULL
   ...> )
   ...> )
   ...> WHERE value = 'restaurant';
304
sqlite>
```

- Number of distinct cafes

```
sqlite> SELECT COUNT(*)
   ...> FROM (
   ...> SELECT DISTINCT *
   ...> FROM (
   ...> SELECT nodes_tags.id, ways_tags.id, nodes_tags.value, ways_tags.value
   ...> FROM nodes_tags
   ...> LEFT JOIN ways_tags USING(id)
   ...> UNION ALL
   ...> SELECT nodes_tags.id, ways_tags.id, nodes_tags.value, ways_tags.value
   ...> FROM ways_tags
   ...> LEFT JOIN nodes_tags USING(id)
   ...> WHERE nodes_tags.id IS NULL
   ...> )
   ...> )
   ...> WHERE value = 'cafe';
123
sqlite>
```

- Average speed limit

```
sqlite> SELECT AVG(SUBSTR(value, 1, 2))
   ...> FROM ways_tags
   ...> WHERE (key = 'maxspeed') & (type = 'regular');
31.426287326593
sqlite>
```

- Maximum speed limit

```
sqlite> SELECT MAX(SUBSTR(value, 1, 2))
   ...> FROM ways_tags
   ...> WHERE (key = 'maxspeed') & (type = 'regular');
79
```

- Minimum speed limit

```
sqlite> SELECT MIN(SUBSTR(value, 1, 2))
   ...> FROM ways_tags
   ...> WHERE (key = 'maxspeed') & (type = 'regular');
10
```

- Number of distinct addresses

```
sqlite> SELECT COUNT(*)
   ...> FROM (
   ...> SELECT DISTINCT *
   ...> FROM (
   ...> SELECT nodes_tags.id, ways_tags.id, nodes_tags.type, ways_tags.type
   ...> FROM nodes_tags
   ...> LEFT JOIN ways_tags USING(id)
   ...> UNION ALL
   ...> SELECT nodes_tags.id, ways_tags.id, nodes_tags.type, ways_tags.type
   ...> FROM ways_tags
   ...> LEFT JOIN nodes_tags USING(id)
   ...> WHERE nodes_tags.id IS NULL
   ...> )
   ...> )
   ...> WHERE type = 'addr';
12100
sqlite>
```

- Number of distinct streets ending in ‘Avenue’

```
sqlite> SELECT COUNT(*)
   ...> FROM (
   ...> SELECT DISTINCT *
   ...> FROM (
   ...> SELECT nodes_tags.value, ways_tags.value
   ...> FROM nodes_tags
   ...> LEFT JOIN ways_tags USING(value)
   ...> UNION ALL
   ...> SELECT nodes_tags.value, ways_tags.value
   ...> FROM ways_tags
   ...> LEFT JOIN nodes_tags USING(value)
   ...> WHERE nodes_tags.id IS NULL
   ...> )
   ...> )
   ...> WHERE value LIKE '% Avenue';
163
sqlite>
```

- Number of distinct streets ending in ‘Street’

```
sqlite> SELECT COUNT(*)
   ...> FROM (
   ...> SELECT DISTINCT *
   ...> FROM (
   ...> SELECT nodes_tags.value, ways_tags.value
   ...> FROM nodes_tags
   ...> LEFT JOIN ways_tags USING(value)
   ...> UNION ALL
   ...> SELECT nodes_tags.value, ways_tags.value
   ...> FROM ways_tags
   ...> LEFT JOIN nodes_tags USING(value)
   ...> WHERE nodes_tags.id IS NULL
   ...> )
   ...> )
   ...> WHERE value LIKE '% Street';
90
sqlite>
```

- Number of distinct streets ending in 'Way'

```
sqlite> SELECT COUNT(*)
   ...> FROM (
   ...> SELECT DISTINCT *
   ...> FROM (
   ...> SELECT nodes_tags.value, ways_tags.value
   ...> FROM nodes_tags
   ...> LEFT JOIN ways_tags USING(value)
   ...> UNION ALL
   ...> SELECT nodes_tags.value, ways_tags.value
   ...> FROM ways_tags
   ...> LEFT JOIN nodes_tags USING(value)
   ...> WHERE nodes_tags.id IS NULL
   ...> )
   ...> )
   ...> WHERE value LIKE '% Way';
41
sqlite>
```

OTHER IDEAS:

It would be remiss to leave out a few words on what could be improved upon with regard to this project. Here I will list a couple ideas that could be explored and their merits and associated difficulties.

- An interesting improvement could revolve around the investigation into speed limits. Specifically, it would be interesting to examine speed limit in proportion to the length of the roads they apply to. Without this factor included in the proper analysis, the

accuracy of our current measure of the “average” speed limit remains imprecise. However, inclusion of such a factor as length of particular roadways seems my particular beyond the knowledge at the moment.

- I would like to imagine a simple way of querying the database for the number of distinct businesses in the area of concern, but after some consideration I concluded that this would be quite complex, requiring the inclusion of a plethora of keywords in the query. Even then, however, it would be unlikely that every necessary keyword would be included. Therefore, I invite explications of easier ways to accomplish this task that I have not yet discovered.

- The problem of what to do with the phone numbers that did not match any of the conceivable formats remains unresolved. One possible way to improve the situation involves those non-phone number values that are website addresses. Specifically, I think it may be possible to find the phone number at the website address, in which case it is conceivable that we might scrape the phone numbers from the relevant websites and insert them in place of the website addresses. Of course, this sounds rather difficult and the efficacy of such an idea remains to be discovered.