Data Wrangling Project

Open Street Map Study

Introduction

The aim of this project is to showcase skills in data wrangling. Data wrangling is the process of extracting, transforming, cleaning, and mapping data from a raw data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.

OpenStreetMap is an open source database of map data built by a community of mappers who contribute and maintain data about roads, places, public transit, and much more, all over the world. OpenStreetMap emphasizes local knowledge, and contributors input data from their communities. Much of the data is entered manually, meaning it is expected to be dirty (erroneous).

Data Wrangling

The data munging process begins with extracting the data in a raw form from the data source, cleaning the raw data, and manipulating the data using algorithms or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use.

The data transformations are applied to distinct entities (e.g. fields, rows, columns, data values etc.) within a data set, and could include such actions as extractions, parsing, joining, standardizing, augmenting, cleansing, consolidating, and filtering to create desired wrangling outputs that can be leveraged downstream.

Map Area

The area chosen is the Upper West Side neighborhood in Manhattan, New York City where the author resides and has local knowledge. The data was surprisingly clean and my mentor reviewing this project should note that the sample database provided was intentionally manually corrupted to test the code submitted.

Specifically, the area is from West 59th Street, Columbus Circle up to West 101st Street (North - South direction) and from the West Side Highway, NY Route 9A to Central Park West (East - West direction).

Data File Size

The data file, ***UpperWestSideFull.osm*** is 78.2MB. A smaller, test data file, ***UpperWestSideTest.osm*** is 10.1MB.

TIGER

The Topologically Integrated Geographic Encoding and Referencing system (TIGER) data, produced by the US Census Bureau, is a public domain data source which has many geographic features. The TIGER/Line files are extracts of selected geographic information, including roads, boundaries, and hydrography features. All of the roads were imported into OpenStreetMap in 2007 and 2008, populating the nearly empty map of the United States.

Data Quality

The data wrangling in this project focuses on data quality. In particular, data quality is determined by the following measures:

- Validity: conforms to a schema
- Accuracy: meets or exceeds an industry standard
- Completeness: breadth of data used. Data is cleaned to limit discarding
- Consistency: matches other data
- Uniformity: matching units

Data Cleaning

A key dimension of the process is to clean any bad data:

- Audit: examine the data to assess the damage
- Causes: identify patterns and the origin of problems
- Plan: determine methods to computationally clean the data
- Strategy: clean the data using programmatic techniques, and use a database for the analysis

Correcting the data involves removing typographical errors, enhancing data, validating data against known entities, changing or mapping data, and standardizing or formatting data.

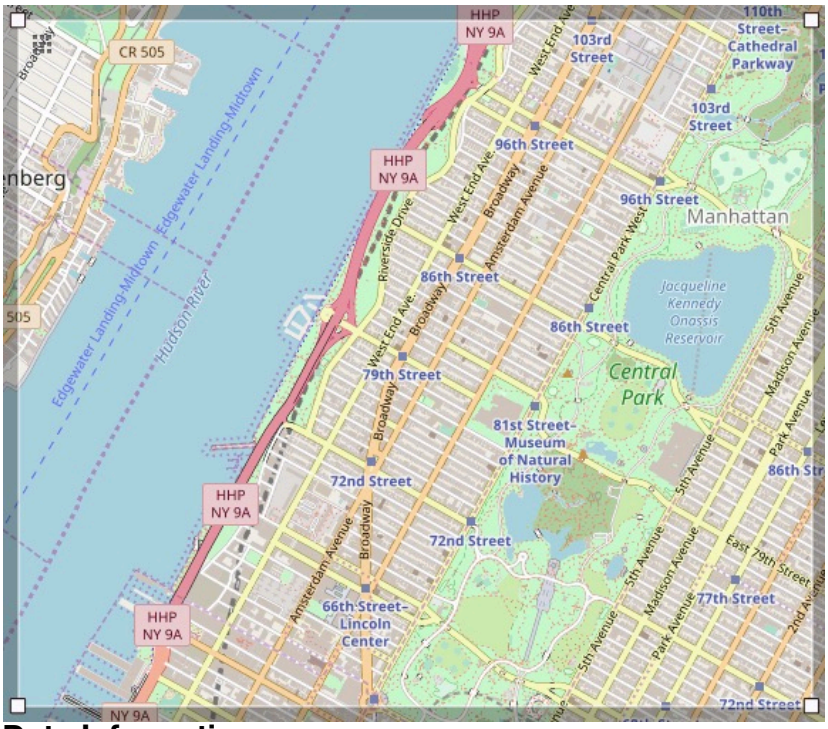
Customization

It should be noted, that the data wrangling employed is customized to the dataset and the results required for a particular project. Each situation will be unique based on the needs of the output.

Project Technology

The map data for the given region is obtained from OpenStreetMap.org in an XML format file. Python programming is used to read in the data, perform the munging, and write out the data to CSV files. Then, the CSV files are imported into a SQLite database where the data analysis is performed.

Map of the Upper West Side in Manhattan, New York



Data Information

Relevant data imported from the raw XML file includes:

– Latitude
– Longitude
– Time stamp
– Key, value pairs identifying roads and places

Data Dictionary

Data quality checks and corrections are applied to the following data:
(listed with the Open Street Map tag keys)

Attribute	Tag Key
Address house number	key = addr:housenumber
Address street	key = addr:street
Address city	key = addr:city
Address state	key = addr:state
Address postal code	key = addr:postcode
Telephone number	key = phone
Email address	key = email
Website URL	key = website or key = url
TIGER info	key = tiger
Name of the establishment	key = name
Type of store	key = shop
Type of building	key = building
Type of amenity	key = amenity
Type of cuisine	key = cuisine
Inscription text	key = inscription

Methodology

Synopsis

The dataset is contained in an XML file. The main function iteratively reads each XML element from the file, processes the element, and writes the parsed data to a CSV file for later import to a database. The central process builds a Python dictionary for the current element in the iteration, and cleans the data. The dictionary is constructed conforming to a defined schema, and each element is validated against the schema before writing it to the CSV file.

An SQL database is created according to the schema, and the data from the CSV files are imported. The data is analyzed issuing SQL commands to the database.

Initial Data Scan

Run a first pass scan over the dataset:

- Identify the problem data
- Determine the **magnitude** of the data issues
- Quantify the data problems using counts

Method

Use reverse logic: The correct data is known. Identify any data that is rejected on correctness. Accuracy is determined by matching against known correct data and implementing data rules using logic and regular expressions.

Python code:
initial_scan.py

Magnitude of the Data Problems

Read in the data. Determine the size of the dataset. Identify problems and the magnitude of the problematic data. The methodology employs regular expressions, lookup lists, and counts.

The idea is to get a handle on how dirty the data is. Perform a count of all the different tags and then compare the problem data to the total number of tags.

Results from the Initial Scan

```
Number of street issues: 55
Total number of streets: 26,222   Percent problems: 0.2%

Number of city issues: 47
Total number of cities: 2,596   Percent problems: 1.8%

Number of state issues: 48
Total number of states: 1,722   Percent problems: 2.8%

Number of zipcode issues: 19
Total number of zipcodes: 25,559   Percent problems: 0.1%

Number of zipcodes outside Upper West Side: 18,869
Total number of zipcodes: 25,559   Percent outside UWS: 73.8%

Number of phone number issues: 13
Total number of phone numbers: 936   Percent problems: 1.4%

Number of email address issues: 0
Total number of email addresses: 133   Percent problems: 0.0%

Number of website URL issues: 1
Total number of websites: 1,564   Percent problems: 0.1%

Number of TIGER issues: 45
Total number of TIGER: 1,951   Percent problems: 2.3%

Total record count: 1,083,410
```

Observations

The data are surprisingly ***clean!*** A surprise because the data are largely entered by hand. I suspect the data were previously cleaned.

Note: The full dataset is used above. The reviewer should note that I have manually corrupted the test dataset contained herein to illustrate the functionality of the code.

Although the magnitude of the problem data is small there are opportunities to remedy several issues.

Corrections:
- Abbreviations
- Spelling

– Punctuation
– Upper / lower case

Data quality checks for:
– Validity
– Accuracy
– Consistency
– Uniformity

Correcting the Problematic Data

Method

During processing, each element is inspected by examining the key and the value of the element tag. The key identifies the type of element currently in process, and the value is directed to the appropriate function for data correction.

Define the Data Problems to be Corrected

The Open Street Map XML file contains several tags. This study utilizes the **`<node>`** and **`<way>`** tags in the XML data source. Inside these tags are children **`<tag>`** tags that are evaluated.

Data not meeting the quality checks are **removed** from the dataset before writing the record to the CSV file.

Process

The strategy employed is to correct the data first, if possible. Then eliminate the bad data that is not fixed.

Data validation of the element ID, tag key, and `<way node>` reference is performed in the *build_dictionary_element_tree* function **before** the tag is sent to the data correction function. A further description is provided in the discussion of the dictionary build (see below).

Data Fields

The following data fields are examined and cleaned:

Streets
Data quality checks for ***validity***, ***consistency*** and ***uniformity***
Methodology: Separate the name of the street from the street description and examine each.
– Lookup list, mapping dictionary, and Python code (dictionaries)

City
Data quality checks for ***accuracy***, ***consistency*** and ***uniformity***
Methodology: Lookup list, mapping dictionary, and Python code (dictionaries)

State
Data quality checks for ***accuracy***, ***consistency*** and ***uniformity***
Methodology: Lookup list, Python code (dictionaries)

Zip Codes
Data quality checks for ***validity***, ***consistency*** and ***uniformity***
Methodology: Lookup list, Python code (dictionaries)

Telephone
Data quality checks for ***validity***, ***consistency*** and ***uniformity***
Methodology: Lookup list, regular expression, Python code (dictionaries)

Email
Data quality checks for ***validity***, ***consistency*** and ***uniformity***
Methodology: Regular expression, lookup list, Python code (dictionaries)

Website
Data quality checks for ***validity***, ***consistency*** and ***uniformity***
Methodology: Regular expression, lookup list, Python code (dictionaries)

TIGER
Data quality checks for ***consistency*** and ***uniformity***
Methodology: Lookup list, Python code (dictionaries)

Other Tag Keys
Data quality checks for ***validity***, ***consistency*** and ***uniformity***
Specific keys:

```
k="addr:housenumber"
k="amenity"
k="name"
k="cuisine"
```

```
k="shop"
k="building"
k="inscription"
```
Methodology: Regular expression, Python code (dictionaries)

Node ID, Way ID and Way Node Reference
Data quality checks for ***validity***
Methodology: Python code

Data Elimination

Corrupt data are removed from the dataset.
Data checks include:
- Null data or whitespace issues
- Text character set and digits
- Format compliance

Programmatic Data Correction and Elimination

The automated process is demonstrated in the routine ***fix_it_demo.py***

Test Dataset

The summary report for the Test dataset is provided below for readers wishing to run the enclosed Python files.

```
SUMMARY

Tag counts:
    Nodes: 36,252
    Ways: 4,268
    Nodes tags: 6,080
    Ways tags: 11,845
    Ways Nodes: 46,990

Eliminated tag counts:
    Tags with bad data values
        Nodes tags voided: 15
        Ways tags voided: 2

    Tags with corrupt ID
        Nodes removed: 2
        Ways removed: 1

    Tags with corrupt keys
        Nodes tags with key problem:  1
        Ways tags with key problem:  1

    Tags with defective reference
        Ways Nodes tags voided: 2

Skipped tag counts
    Node child tags skipped: 3,495
    Way child tags skipped: 11,620
    Total tags skipped: 15,115

Total non-node or non-way count: 91,809

Total elements processed: 132,332
```

Machine Automation

The programmatic routines below are used to perform the data wrangling:

- ***fix_it.py***
- ***element_to_dictionary.py***
- ***main_process.py***

The output are the validated (against the schema) CSV files containing the parsed dataset.

NOTE:

The modules ***fix_it.py*** and ***element_to_dictionary.py*** **cannot** be executed independently. They are **included** in *main_process.py*.
To create the CSV files, run ***main_process.py***.

The module ***fix_it_demo.py*** executes a demonstration of the programmatic data cleaning implemented.

Python code:
fix_it_demo.py
fix_it.py

Extract, Transform, and Load

The process for correcting or eliminating data values is explained above (***fix_it_demo.py***). What follows is a description of the surrounding methodology that is the data engine.

To review: the Open Street Map data is provided in an XML document format file. That file is read, the data parsed, cleaned, and stored into tabular data structures. It is then written into CSV format files. Later, the CSV file data are imported into an SQL database for data analysis.

Extracting the Data

XML is an inherently hierarchical data format, and the most natural way to represent it is with a tree. Python's ElementTree has two classes for this purpose -- ElementTree represents the whole XML document as a tree, and Element represents a single node in this tree. Interactions with the whole document (reading and writing to/from files) are usually done on the ElementTree level. Interactions with a single XML element and its sub-elements are done on the Element level.

The ElementTree wrapper type adds code to load XML files as trees of Element objects, and save them back again. The Element type is a simple but flexible container **object**, designed to store hierarchical data structures, such as simplified XML infosets, in memory.

Referring to the file ***main_process.py*** (see the second cell below), the ***get_element_tree*** function utilizes the ElementTree ***iterparse*** function to iteratively step through each top level XML element and read in sections of the XML file as a tree of the element, for example:

```
...                 -- element tree
<node>              -- top level XML element
  <tag ... />       -- children tags (sub-elements)
</node>
...
```

The element tree is returned as a **generator** and the memory space of the block is cleared. This allows huge files to be incrementally processed without saving the entire file in memory.

Transforming the Data

Referring to the file ***element_to_dictionary.py*** (see the cell below), the ***build_dictionary_element_tree*** function treats `<node>` element trees separately from `<way>` element trees. For the `<node>` elements, a dictionary of the `<node>` attributes is created. For each of the child `<tag>` tags, the tag value is sent to the ***fixer*** function (module ***fix_it.py***) for correction or elimination. A list of the node tags is created.

Similarly for the `<way>` elements, a dictionary of the `<way>` attributes is created and the child `<tag>` values are sent to the ***fixer*** function. Lists of way tags and way-nodes are created.

The ***build_dictionary_element_tree*** function returns a dictionary of the attributes and tags for the `<node>` or `<way>` element tree.

Loading the Data

Back to the file ***main_process.py*** (see the second cell below), the returned dictionaries are validated using the Cerberus library against the database schema. CSV files are created and each element and tag are written as a row in the appropriate CSV file.

Detailed Description of the *build_dictionary_element_tree* Function

The function takes as input an iterparse Element object (element tree) and returns a dictionary.

Python code:
element_to_dictionary.py

Data Validation

The function validates the element attribute 'id' is digits only. If the 'id' is null or not an integer, the element is eliminated from the dataset. The 'id' must be valid because it serves as the primary key in the SQL database.

For each tag, the key is validated to contain correct characters. The tags with keys that contain incorrect characters are removed from the dataset. This approach improves run-time performance -- there is no need to send tags with invalid keys to the data correction engine.

If the tag is a `<way node>` , the reference attribute 'ref' is validated to contain digits only. If the reference is null or not an integer, the tag is eliminated from the dataset. The 'ref' must be valid because it serves as a foreign key in the SQL database.

Tag Processing

If the element top level tag is `<node>` :

The dictionary returned has the format {"node": .., "node_tags": ...}

The "node" item is a key and its value holds a dictionary of the following top level node attributes (dictionary within a dictionary):
– id
– user
– uid
– version
– lat
– lon
– timestamp
– changeset

All other attributes are ignored.

The "node_tags" item is a key and its value holds a list of dictionaries, one per secondary tag. Secondary tags are child tags of the node which have the tag name/type: `<tag>` . Each dictionary has the following fields from the secondary tag attributes:
– id: the top level node id attribute value
– key: the full tag "k" attribute value if no colon is present, or the characters after the first colon if one or more colons exist
– value: the tag "v" attribute value
– type: either the characters before the colon in the tag "k" value or "regular" if a colon is not present

Additionally,
– if the tag "k" attribute contains problematic characters, the tag is ignored
– if the tag "k" attribute contains a ":" the characters before the ":" are set as the tag type and characters after the ":" are set as the tag key
– if there are additional ":" in the "k" attribute they are ignored and kept as part of the tag key. For example:

```
<tag k="addr:street:name" v="Lincoln"/>
```

is turned into

```
{'id': 12345, 'key': 'street:name', 'value': 'Lincoln', 'type': 'addr'}
```

– If a node has no secondary tags then the "node_tags" field contains an empty list

The final return value for a "node" element looks like:

```
{'node': {'id': 757860928, 'user': 'uboot', 'uid': 26299, 'version': '2',
          'lat': 41.9747374, 'lon': -87.6920102, 'timestamp': '2010-07-22T16:16:51Z',
          'changeset': 5288876},
 'node_tags': [{'id': 757860928, 'key': 'amenity', 'value': 'fast_food', 'type': 'regular'},
               {'id': 757860928, 'key': 'cuisine', 'value': 'sausage', 'type': 'regular'},
               {'id': 757860928, 'key': 'name', 'value': "Shelly's Tasty Freeze", 'type': 'regular'}
               ]
 }
```

If the element top level tag is `<way>`:

The dictionary has the format {"way": ..., "way_tags": ..., "way_nodes": ...}

The "way" item is a key and its value holds a dictionary of the following top level way attributes (dictionary of dictionaries):
– id
– user
– uid
– version
– timestamp
– changeset

All other attributes are ignored. The "way_tags" item is a key and its value holds a list of dictionaries, following the exact same rules as for "node_tags".

Additionally, the returned dictionary has an item "way_nodes". "way_nodes" is a key and its value holds a list of dictionaries, one for each "nd" child tag. Each dictionary has the fields:
– id: the top level way element id
– node_id: the ref attribute value of the `<nd>` tag
– position: the index starting at 0 of the `<nd>` tag i.e. what order the `<nd>` tag appears within the way element

The final return value for a "way" element looks like:

```
{'way': {'id': 209809850, 'user': 'chicago-buildings', 'uid': 674454, 'version': '1',
         'timestamp': '2013-03-13T15:58:04Z', 'changeset': 15353317},
 'way_nodes': [{'id': 209809850, 'node_id': 2199822281, 'position': 0},
               {'id': 209809850, 'node_id': 2199822390, 'position': 1},
               {'id': 209809850, 'node_id': 2199822392, 'position': 2},
               {'id': 209809850, 'node_id': 2199822369, 'position': 3},
               {'id': 209809850, 'node_id': 2199822370, 'position': 4},
               {'id': 209809850, 'node_id': 2199822284, 'position': 5},
               {'id': 209809850, 'node_id': 2199822281, 'position': 6}
               ],
 'way_tags': [{'id': 209809850, 'key': 'housenumber', 'type': 'addr', 'value': '1412'},
              {'id': 209809850, 'key': 'street', 'type': 'addr', 'value': 'West Lexington St.'},
              {'id': 209809850, 'key': 'street:name', 'type': 'addr', 'value': 'Lexington'},
              {'id': 209809850, 'key': 'street:prefix', 'type': 'addr', 'value': 'West'},
              {'id': 209809850, 'key': 'street:type', 'type': 'addr', 'value': 'Street'},
              {'id': 209809850, 'key': 'building', 'type': 'regular', 'value': 'yes'}
              ]
 }
```

Main Process

The function ***process_xml_elements*** iteratively parses the XML element tree, calls the function to build a dictionary, validates, and writes the cleaned data to CSV files.

Python code:
main_process.py

Data Wrangling Results

Summary

```
Tag counts:
    Nodes: 289,145
    Ways: 38,744
    Nodes tags: 44,335
    Ways tags: 82,315
    Ways Nodes: 391,360

Eliminated tag counts:
    Tags with bad data values
        Nodes tags voided: 23
        Ways tags voided: 4

    Tags with corrupt ID
        Nodes removed: 0
        Ways removed: 0

    Tags with corrupt keys
        Nodes tags with key problem:  0
        Ways tags with key problem:  0

    Tags with defective reference
        Ways Nodes tags voided: 0

Skipped tag counts
    Node child tags skipped: 21,717
    Way child tags skipped: 105,948
    Total tags skipped: 127,665

Total non-node or non-way count: 755,521

Total elements processed: 1,083,410
```

Details

```
CONSOLIDATED DETAILS OF DATA CORRECTIONS AND ELIMINATIONS

Streets fixed: 61
Number of street issues: 13 streets removed from dataset

Cities fixed: 52
Number of cities not in NYC: 41
Number of city problems: 0 cities removed from dataset

States fixed: 4
Number of state issues: 30
States outside NY:
('NJ', 30)
Number of state problems: 0 states removed from dataset

Zip codes fixed: 17
Number of zipcode issues: 2 zipcodes removed from dataset

Phones fixed: 249
Number of phone number issues: 11 phone numbers removed from dataset

Number of email address issues: 0 emails removed from dataset

Number of website URL issues: 1 websites removed from dataset

Tiger no's fixed: 31
Number of TIGER issues: 0 TIGER tags removed from dataset
(TIGER is not ['yes', 'no', 'aerial'])

House numbers fixed: 0

Cuisine names fixed: 654

Number of other value issues:  0 tags removed from dataset

<nodes><tags> and <ways><tags> corrupted keys: 0 tags removed from dataset

Number of <nodes> eliminated:  0 nodes removed from dataset

Number of <ways> eliminated:  0 ways removed from dataset

Number of <ways nodes> eliminated:  0 ways nodes removed from dataset
```

Test Dataset

The summary report for the Test dataset is provided below for readers wishing to run the enclosed Python files.

```
SUMMARY

Tag counts:
    Nodes: 36,252
    Ways: 4,268
    Nodes tags: 6,080
    Ways tags: 11,845
    Ways Nodes: 46,990

Eliminated tag counts:
    Tags with bad data values
        Nodes tags voided: 15
        Ways tags voided: 2

    Tags with corrupt ID
        Nodes removed: 2
        Ways removed: 1

    Tags with corrupt keys
        Nodes tags with key problem:  1
        Ways tags with key problem:  1

    Tags with defective reference
        Ways Nodes tags voided: 2

Skipped tag counts
    Node child tags skipped: 3,495
    Way child tags skipped: 11,620
    Total tags skipped: 15,115

Total non-node or non-way count: 91,809

Total elements processed: 132,332
```

Cerberus Data Validation

Referring to ***main_process.py*** above, during the parsing process, the XML tag data for each XML element tree, is sanitized and stored into a Python dictionary. This dictionary, and the Python schema, are sent to the Cerberus validator to confirm the dictionary created represents the desired database schema. Only after confirmation, is the data written to the CSV file.

CSV Files

The original data from the Open Street Map XML file is parsed, corrected where possible or eliminated (data cleaning), and saved into CSV files for importing into an SQL database.

First, checks are performed to validate the cleaning process. The number of records in the CSV files are compared against the tag counts output by the Python processing. If the counts match, this validates the CSV files contain only the correct data with the dirty data removed.

The function ***csv_row_count()*** in the code below, performs the check.

Python code:
xml_csv_validation_routines.py

CSV File Record Counts

```
Node number of rows: 289,145
Node tags number of rows: 44,335

Way number of rows: 38,744
Way tags number of rows: 82,315
Way Node number of rows: 391,360
```

Count the XML File Elements

The number of different tags in the original XML file are counted. These numbers are compared against the number of records in the CSV files. The expectation is the number of records in the CSV files is **less** than the number in the XML file due to elimination of dirty data. Further, when the number of bad data is tallied, the counts between the XML and CSV files are expected to reconcile.

The function ***count_xml_tags()*** in the code below, performs the XML check.

Python code:
xml_csv_validation_routines.py

XML File Tag Counts

```
 Nodes:  289,145
 Ways:  38,744
 Ways Nodes:  391,360

 Nodes tags:  44,358
 Ways tags:  82,319

Total sum of XML tags processed: 1,083,410

------------------------------

XML PROBLEM TAGS

Node ID problem:  0
Way ID problem:  0
Way Node problem:  0
Node tag problem:  0
Way tag problem:  0

Node tag key problem:  0
Way tag key problem:  0
```

Reconcile the Tag Counts Between XML and CSV Files

The table below summarizes the comparison. The table confirms the count differences in the files equals the number of bad data eliminated.

This validates that the problem data is eliminated from the CSV files -- which are used to import the data into the database. Examine the XML tag data and reconcile with the CSV record data. The difference between them is expected to equal the number of problem data discarded.

The function ***make_table()*** in the code below creates the reconciled table.

Python code:
xml_csv_validation_routines.py

Table of Record Count Differences

	XML Tag Count	Eliminated XML Tags	CSV Record Count	Difference is Tags Eliminated
<node> <tag>	44,358	0	44,335	23
<way> <tag>	82,319	0	82,315	4
<node>	289,145	0	289,145	0
<way>	38,744	0	38,744	0
<way node>	391,360	0	391,360	0

Note two points in the above table:

- The CSV files contain **less** records than the XML file confirming certain problematic data was removed
- The differences correlate with the number of bad data tags reported by the program ***main_process.py***

Summary of Data Reconciliation for the Test Database

```
SUMMARY

Tag counts:
    Nodes: 36,252
    Ways: 4,268
    Nodes tags: 6,080
    Ways tags: 11,845
    Ways Nodes: 46,990

Eliminated tag counts:
    Tags with bad data values
        Nodes tags voided: 15
        Ways tags voided: 2

    Tags with corrupt ID
        Nodes removed: 2
        Ways removed: 1

    Tags with corrupt keys
        Nodes tags with key problem:  1
        Ways tags with key problem:  1

    Tags with defective reference
        Ways Nodes tags voided: 2

Skipped tag counts
    Node child tags skipped: 3,495
    Way child tags skipped: 11,620
    Total tags skipped: 15,115

Total non-node or non-way count: 91,809

Total elements processed: 132,332
```

```
CSV FILE RECORD COUNTS

Node number of rows: 36,252
Node tags number of rows: 6,080

Way number of rows: 4,268
Way tags number of rows: 11,845
Way Node number of rows: 46,990
```

```
XML FILE TAG COUNTS

• Nodes:  36,254
• Ways:  4,269
• Ways Nodes:  47,001

• Nodes tags:  6,101
• Ways tags:  11,848

Total sum of XML tags processed: 132,332

XML PROBLEM TAGS

Node ID problem:  2
Way ID problem:  1
Way Node problem:  11
Node tag problem:  6
Way tag problem:  1

Node tag key problem:  1
Way tag key problem:  1
```

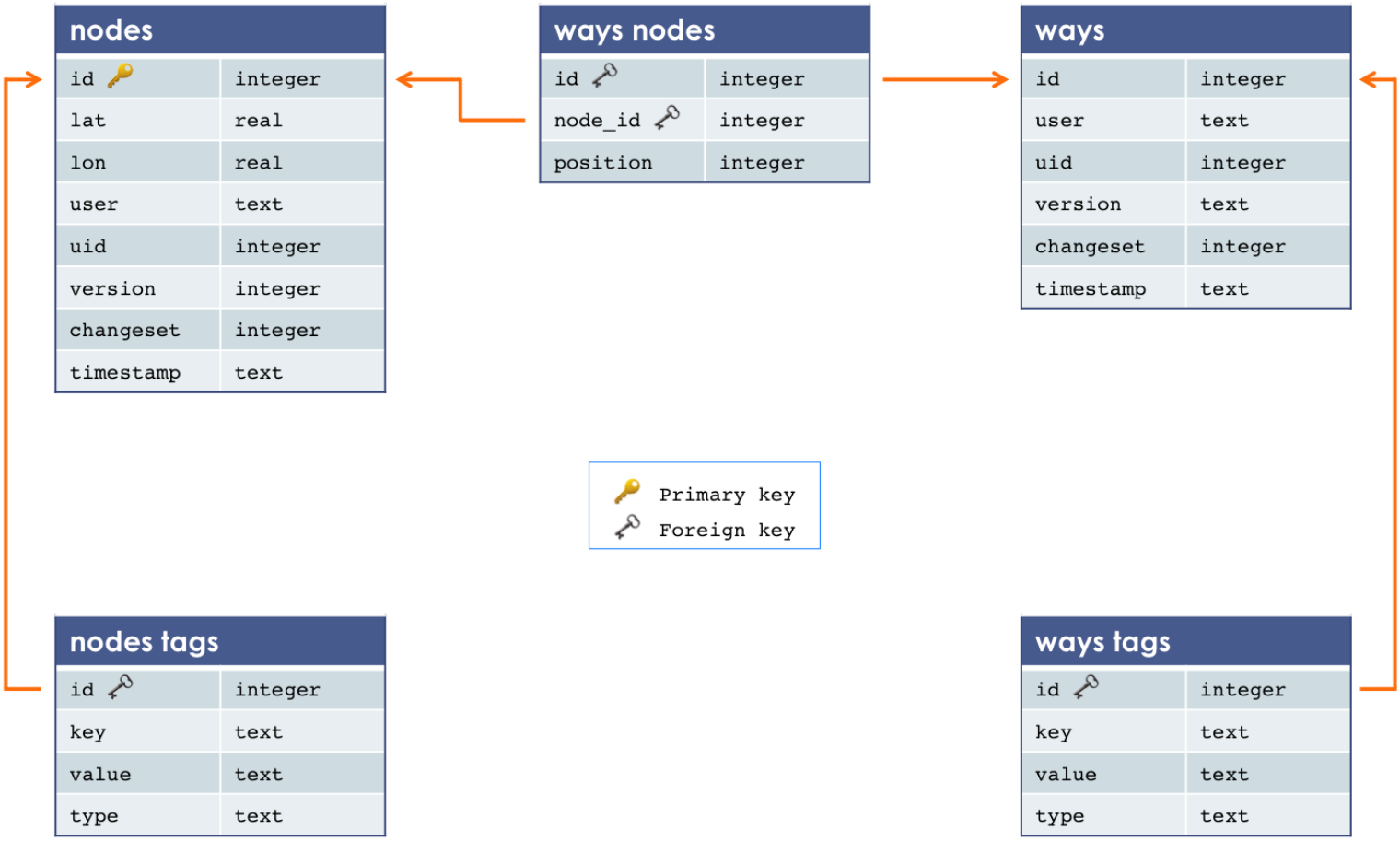
TABLE OF RECORD COUNT DIFFERENCES

	XML Tag Count	Eliminated XML Tags	CSV Record Count	Difference is Tags Eliminated
<node> <tag>	6,101	6	6,080	15
<way> <tag>	11,848	1	11,845	2
<node>	36,254	2	36,252	0
<way>	4,269	1	4,268	0
<way node>	47,001	11	46,990	0

Database Schema

The data in the CSV files are imported into an SQLite database. The database and tables are created according to the ***data_wrangling_schema.sql*** schema specification. The **'id'** fields in the tables are used as the database reference keys.

Database Diagram



For Python processing purposes, the SQL schema is translated to a Python dictionary structure -- see the ***db_schema.py*** module.

Database Creation

The SQLite database is created and the tables are generated as per the schema. This is performed programmatically in Python -- see the function ***create_database()*** in the module below.

Python code:
database_routines.py

CSV File Import

Data in the CSV files are imported into the database tables. The names of the columns are specified and the **abort** statement is applied to flag data insertion errors. The number of records imported to each table is recorded to double-check the counts are consistent.

If an applicable constraint violation occurs, the ABORT resolution algorithm aborts the current SQL statement with an SQLITE_CONSTRAINT error and backs out any changes made by the current SQL statement; but changes caused by prior SQL statements within the same transaction are preserved and the transaction remains active.

See the function ***read_csv_files()*** in the module below.

Python code:
database_routines.py

```
Nodes written to db...
Nodes number of rows: 289,145

Nodes Tags written to db...
Nodes Tags number of rows: 44,335

Ways written to db...
Ways number of rows: 38,744

Ways Tags written to db...
Ways Tags number of rows: 82,315

Ways Nodes written to db...
Ways Nodes number of rows: 391,360
```

Validate the number of rows in each table

The number of rows in each database table is counted. These numbers are compared with the number of rows in the CSV files and the number of correct data output by the program ***main_process.py***.

See the function ***count_rows*** in the module below.

Python code:
database_routines.py

```
Count the number of rows in each table:

Nodes: 289,145
Nodes Tags: 44,335
Ways: 38,744
Ways Tags: 82,315
Ways Nodes: 391,360
```

The above check validates the amount of data loaded into the database tables are correct.

Exploring the Data

Create Consolidated Tables

This project requires information from both the `<nodes>` and the `<ways>` tables because data of interest is located in both. A `<node>` is a point on the map and `<ways>` are paths connecting the nodes.

The strategy is to break complicated database queries into smaller pieces that are simpler and more reliable to execute. First, consolidate the required data into new tables, and then execute the retrieval queries on the new tables.

Create a new table to consolidate `<node>` `<tags>` and `<way>` `<tags>`, and similarly another for `<nodes>` and `<ways>`. Note that latitude and longitude data only occur in `<nodes>`. ID, user, and timestamp data is included in both the `<nodes>` and `<ways>` tables. The NULL statement is used as a placeholder for the latitude and longitude columns in the `<ways>` table.

Refer to the function ***consolidated_tables()*** in the module below.

Python code:
database_routines.py

Database Queries

SQL queries are run to provide a sense of the data coverage and quality.

- Query the age of the data using the timestamp
- Count the zip codes
- Find all the burger places on the Upper West Side
- Search for book stores on the Upper West Side
- Look up the store 'RadioShack' on the Upper West Side which was closed
- A surprise, interesting find -- see the sections below

Python code:
database_queries.py

SQL Query Statements

```
query_1 = "SELECT timestamp, COUNT(timestamp), user FROM nodes_union_ways
         GROUP BY timestamp ORDER BY timestamp ASC LIMIT 10;"

query_2 = "SELECT value, COUNT(value) as Number FROM union_all_tags
         WHERE key = 'postcode'
         GROUP BY value ORDER BY Number DESC LIMIT 10;"

query_3 = "SELECT union_all_tags.id, key, value, timestamp FROM union_all_tags
         INNER JOIN nodes_union_ways
         ON nodes_union_ways.id = union_all_tags.id
         WHERE union_all_tags.id IN (SELECT id FROM union_all_tags
                                     WHERE key = 'cuisine' AND value = 'Burger'
                                     INTERSECT
                                     SELECT id FROM union_all_tags
                                     WHERE key = 'postcode' AND value IN ('10023', '10024', '10025')
                                     );"

query_4 = "SELECT union_all_tags.id, key, value, timestamp FROM union_all_tags
         INNER JOIN nodes_union_ways
         ON nodes_union_ways.id = union_all_tags.id
         WHERE union_all_tags.id IN (SELECT id FROM union_all_tags
                                     WHERE key = 'shop' AND value = 'books'
                                     INTERSECT
                                     SELECT id FROM union_all_tags
                                     WHERE key = 'postcode' AND value IN ('10023', '10024', '10025')
                                     );"

query_5 = "SELECT union_all_tags.id, key, value, timestamp FROM union_all_tags
         INNER JOIN nodes_union_ways
         ON nodes_union_ways.id = union_all_tags.id
         WHERE union_all_tags.id IN (SELECT id FROM union_all_tags
                                     WHERE key = 'name' AND value IN ('RadioShack', 'Radio Shack', 'Radioshack')
                                     INTERSECT
                                     SELECT id FROM union_all_tags
                                     WHERE key = 'postcode' AND value IN ('10023', '10024', '10025')
                                     );"

query_6 = "SELECT union_all_tags.id, key, value,
           CASE WHEN lat IS NOT NULL
           THEN lat
           ELSE ' '
           END,
           CASE WHEN lon IS NOT NULL
           THEN lon
           ELSE ' '
           END
         FROM union_all_tags
         INNER JOIN nodes_union_ways
         ON nodes_union_ways.id = union_all_tags.id
         WHERE union_all_tags.id IN (SELECT id FROM union_all_tags
                                     WHERE key IN ('inscription_1', 'inscription_2', 'inscription_date',
                                                   'nrhp:inscription_date')
                                     );"
```

SQL Query Output

Dates of data entry

  Provides the Age of the Data
  Oldest date first, in descending order

Timestamp	Count	Username
2007-10-08T20:32:49Z	16	KindredCoda
2007-10-08T20:39:28Z	17	KindredCoda
2007-10-08T20:39:37Z	18	KindredCoda
2007-10-08T20:43:15Z	2	KindredCoda
2007-10-08T20:57:15Z	1	KindredCoda
2007-12-15T08:33:56Z	27	KindredCoda
2008-01-27T22:20:36Z	8	Ebenezer
2008-02-10T22:46:34Z	2	jlert
2008-02-10T22:48:59Z	1	jlert
2008-02-10T23:50:35Z	3	jlert

Zip code

  List of the Zip Codes
  ** indicates Upper West Side

Zip code	Count
10025 **	2489
10024 **	2419
10019	1868
10128	1754
10029	1734
10021	1732
10065	1721
10028	1720
10027	1714
10023 **	1703

Burger Joints on the Upper West Side

ID	Key	Value	Timestamp
1821015975	city	New York	2016-05-12T12:33:08Z
1821015975	housenumber	2315	2016-05-12T12:33:08Z
1821015975	postcode	10024	2016-05-12T12:33:08Z
1821015975	state	NY	2016-05-12T12:33:08Z
1821015975	street	Broadway	2016-05-12T12:33:08Z
1821015975	amenity	restaurant	2016-05-12T12:33:08Z
1821015975	cuisine	Burger	2016-05-12T12:33:08Z
1821015975	name	5 Napkin Burger	2016-05-12T12:33:08Z
1821015975	phone	+1 212 333 4488	2016-05-12T12:33:08Z
1821015975	website	http://5napkinburger.com/upper-west-side-new-york	2016-05-12T12:33:08Z
1888172061	housenumber	366	2017-12-28T00:01:52Z
1888172061	postcode	10024	2017-12-28T00:01:52Z
1888172061	street	Columbus Avenue	2017-12-28T00:01:52Z
1888172061	amenity	restaurant	2017-12-28T00:01:52Z
1888172061	cuisine	Burger	2017-12-28T00:01:52Z
1888172061	name	Shake Shack	2017-12-28T00:01:52Z
1888172061	website	http://www.shakeshack.com/location/upper-west-side/	2017-12-28T00:01:52Z
2312065727	city	New York	2013-05-19T22:01:55Z
2312065727	housenumber	2847	2013-05-19T22:01:55Z
2312065727	postcode	10025	2013-05-19T22:01:55Z
2312065727	state	NY	2013-05-19T22:01:55Z
2312065727	street	Broadway	2013-05-19T22:01:55Z
2312065727	amenity	fast_food	2013-05-19T22:01:55Z
2312065727	cuisine	Burger	2013-05-19T22:01:55Z
2312065727	name	Five Guys	2013-05-19T22:01:55Z
2409398831	city	New York	2014-03-31T22:00:43Z
2409398831	housenumber	517	2014-03-31T22:00:43Z
2409398831	postcode	10024	2014-03-31T22:00:43Z
2409398831	state	NY	2014-03-31T22:00:43Z
2409398831	street	Columbus Avenue	2014-03-31T22:00:43Z
2409398831	amenity	fast_food	2014-03-31T22:00:43Z
2409398831	cuisine	Burger	2014-03-31T22:00:43Z
2409398831	name	Jackson Hole	2014-03-31T22:00:43Z
2409398831	website	http://www.jacksonholeburgers.com/	2014-03-31T22:00:43Z
2443892135	city	New York	2013-09-03T02:36:23Z
2443892135	housenumber	2049	2013-09-03T02:36:23Z
2443892135	postcode	10023	2013-09-03T02:36:23Z

```
2443892135        state             NY                                                    2013-09-03T02:36:23Z
2443892135        street            Broadway                                              2013-09-03T02:36:23Z
2443892135        amenity           fast_food                                             2013-09-03T02:36:23Z
2443892135        cuisine           Burger                                                2013-09-03T02:36:23Z
2443892135        name              McDonald's                                            2013-09-03T02:36:23Z
2443892135        website           http://www.mcnewyork.com/2292                         2013-09-03T02:36:23Z

2707983903        city              New York                                              2014-04-03T19:52:57Z
2707983903        housenumber       2850                                                  2014-04-03T19:52:57Z
2707983903        postcode          10025                                                 2014-04-03T19:52:57Z
2707983903        street            Broadway                                              2014-04-03T19:52:57Z
2707983903        amenity           restaurant                                            2014-04-03T19:52:57Z
2707983903        cuisine           Burger                                                2014-04-03T19:52:57Z
2707983903        name              Mel's Burger Bar                                      2014-04-03T19:52:57Z

2759235699        city              New York                                              2014-04-01T19:39:06Z
2759235699        housenumber       654                                                   2014-04-01T19:39:06Z
2759235699        postcode          10025                                                 2014-04-01T19:39:06Z
2759235699        state             NY                                                    2014-04-01T19:39:06Z
2759235699        street            Amsterdam Avenue                                      2014-04-01T19:39:06Z
2759235699        amenity           restaurant                                            2014-04-01T19:39:06Z
2759235699        cuisine           Burger                                                2014-04-01T19:39:06Z
2759235699        name              Amsterdam Burger Co                                   2014-04-01T19:39:06Z

269318133         city              New York                                              2017-12-28T12:54:14Z
269318133         housenumber       422                                                   2017-12-28T12:54:14Z
269318133         postcode          10024                                                 2017-12-28T12:54:14Z
269318133         state             NY                                                    2017-12-28T12:54:14Z
269318133         street            Amsterdam Avenue                                      2017-12-28T12:54:14Z
269318133         amenity           restaurant                                            2017-12-28T12:54:14Z
269318133         building          yes                                                   2017-12-28T12:54:14Z
269318133         cuisine           Burger                                                2017-12-28T12:54:14Z
269318133         name              Island Burgers and Shakes                             2017-12-28T12:54:14Z
269318133         website           http://www.islandburgersandshakes.com/upwest.htm      2017-12-28T12:54:14Z

270985138         city              New York                                              2015-11-26T19:09:34Z
270985138         housenumber       2549                                                  2015-11-26T19:09:34Z
270985138         postcode          10025                                                 2015-11-26T19:09:34Z
270985138         state             NY                                                    2015-11-26T19:09:34Z
270985138         street            Broadway                                              2015-11-26T19:09:34Z
270985138         amenity           fast_food                                             2015-11-26T19:09:34Z
270985138         building          yes                                                   2015-11-26T19:09:34Z
270985138         cuisine           Burger                                                2015-11-26T19:09:34Z
270985138         name              McDonald's                                            2015-11-26T19:09:34Z
270985138         website           http://www.mcnewyork.com/2002                         2015-11-26T19:09:34Z

270991644         city              New York                                              2014-04-01T20:54:14Z
270991644         housenumber       726                                                   2014-04-01T20:54:14Z
270991644         postcode          10025                                                 2014-04-01T20:54:14Z
270991644         state             NY                                                    2014-04-01T20:54:14Z
270991644         street            Amsterdam Avenue                                      2014-04-01T20:54:14Z
270991644         amenity           restaurant                                            2014-04-01T20:54:14Z
270991644         building          yes                                                   2014-04-01T20:54:14Z
270991644         cuisine           Burger                                                2014-04-01T20:54:14Z
270991644         name              Gotham Burger Company                                 2014-04-01T20:54:14Z
270991644         website           http://www.gothamburgers.com/                         2014-04-01T20:54:14Z

Bookshops on the Upper West Side
--------------------------------

ID                Key               Value                                                 Timestamp
--                ---               -----                                                 ---------
2279017951        city              New York                                              2013-04-24T01:59:05Z
2279017951        housenumber       2879                                                  2013-04-24T01:59:05Z
2279017951        postcode          10025                                                 2013-04-24T01:59:05Z
2279017951        state             NY                                                    2013-04-24T01:59:05Z
2279017951        street            Broadway                                              2013-04-24T01:59:05Z
2279017951        name              Bank Street Bookstore                                 2013-04-24T01:59:05Z
2279017951        shop              books                                                 2013-04-24T01:59:05Z
2279017951        website           http://www.bankstreetbooks.com/                       2013-04-24T01:59:05Z

2707989850        city              New York                                              2017-12-08T21:32:48Z
2707989850        housenumber       536                                                   2017-12-08T21:32:48Z
2707989850        postcode          10025                                                 2017-12-08T21:32:48Z
2707989850        street            West 112th Street                                     2017-12-08T21:32:48Z
2707989850        name              Book Culture                                          2017-12-08T21:32:48Z
2707989850        phone             +1-212-865-1588                                       2017-12-08T21:32:48Z
2707989850        shop              books                                                 2017-12-08T21:32:48Z
2707989850        website           http://www.bookculture.com/                           2017-12-08T21:32:48Z

2758684017        city              New York                                              2014-04-01T16:54:40Z
2758684017        housenumber       2289                                                  2014-04-01T16:54:40Z
2758684017        postcode          10024                                                 2014-04-01T16:54:40Z
2758684017        state             NY                                                    2014-04-01T16:54:40Z
```

2758684017	street	Broadway	2014-04-01T16:54:40Z
2758684017	name	Barnes & Noble	2014-04-01T16:54:40Z
2758684017	shop	books	2014-04-01T16:54:40Z
2796448532	city	New York	2017-12-08T21:34:39Z
2796448532	housenumber	2915	2017-12-08T21:34:39Z
2796448532	postcode	10025	2017-12-08T21:34:39Z
2796448532	state	NY	2017-12-08T21:34:39Z
2796448532	street	Broadway	2017-12-08T21:34:39Z
2796448532	name	Book Culture	2017-12-08T21:34:39Z
2796448532	phone	+1-646-403-3000	2017-12-08T21:34:39Z
2796448532	shop	books	2017-12-08T21:34:39Z
2796448532	website	http://www.bookculture.com/	2017-12-08T21:34:39Z
4327964095	city	New York	2017-12-08T21:35:27Z
4327964095	housenumber	450	2017-12-08T21:35:27Z
4327964095	postcode	10024	2017-12-08T21:35:27Z
4327964095	state	NY	2017-12-08T21:35:27Z
4327964095	street	Columbus Avenue	2017-12-08T21:35:27Z
4327964095	name	Book Culture	2017-12-08T21:35:27Z
4327964095	phone	+1-212-595-1962	2017-12-08T21:35:27Z
4327964095	shop	books	2017-12-08T21:35:27Z
4327964095	website	www.bookculture.com	2017-12-08T21:35:27Z

RadioShack

ID	Key	Value	Timestamp
2758620399	city	New York	2014-04-01T16:25:19Z
2758620399	housenumber	2268	2014-04-01T16:25:19Z
2758620399	postcode	10024	2014-04-01T16:25:19Z
2758620399	state	NY	2014-04-01T16:25:19Z
2758620399	street	Broadway	2014-04-01T16:25:19Z
2758620399	name	RadioShack	2014-04-01T16:25:19Z
2758620399	shop	electronics	2014-04-01T16:25:19Z

Inscription

ID	Key	Value	Latitude	Longitude
3126149298	inscription_1	Dedicated to the indomitable spirit of the sled dogs that	40.7699908	-73.9710028
3126149298	inscription_1	relayed antitoxin six hundred miles over rough ice, across	40.7699908	-73.9710028
3126149298	inscription_1	treacherous waters, through Arctic blizzards from Nenana	40.7699908	-73.9710028
3126149298	inscription_1	to the relief of stricken Nome in the Winter of 1925	40.7699908	-73.9710028
3126149298	inscription_2	Endurance · Fidelity · Intelligence	40.7699908	-73.9710028
266940614	housenumber	15		
266940614	postcode	10128		
266940614	street	East 96th Street		
266940614	building	house		
266940614	name	Lucy Drexel Dahlgren House		
266940614	inscription_date	1989		
266940626	housenumber	7		
266940626	postcode	10029		
266940626	street	East 96th Street		
266940626	building	house		
266940626	name	Ogden Codman, Jr. House		
266940626	inscription_date	1967		

Surprise!

I found a statue in Central Park that was unbeknown to me, with its inscription -- to be revealed ahead.

Query Results -- Part I

Assess the data quality by coverage, accuracy, and age.

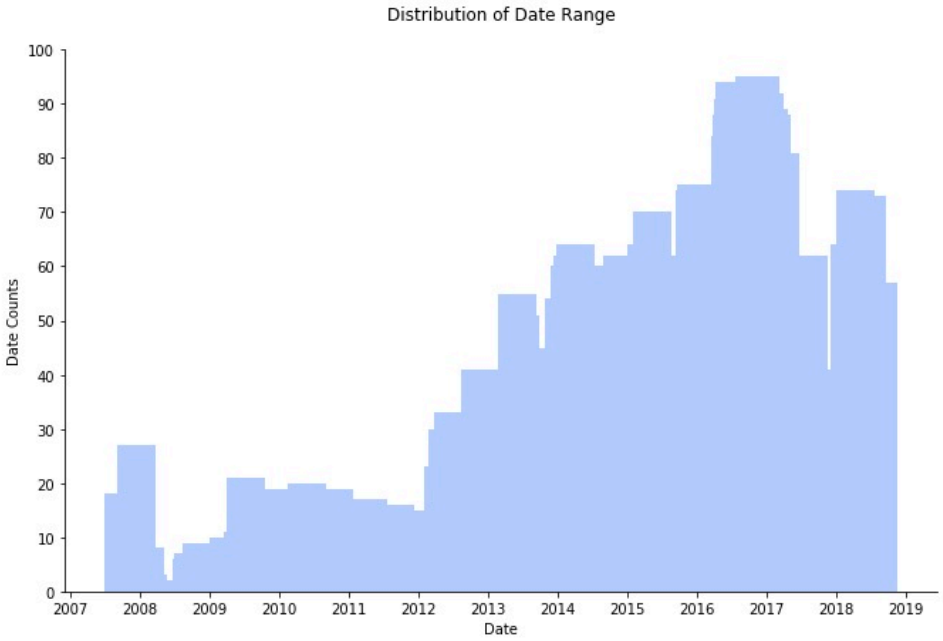
- Most of the data is from zip codes 10023, 10024, and 10025 -- the Upper West Side
- Coverage of amenities and cuisines is good
- The list of places is incomplete as seen by the burger and book store queries -- missing data is a problem.
- The 'RadioShack' store is closed yet continues to be listed -- outdated data is erroneous

Plot the distribution of the age of the data

This plot answers the question: "How old is the data?"
In mapping applications, the age of the data is **important** because places change constantly. A map with old data is not useful in a rapidly changing metropolis such as New York.

Refer to the function ***plot_dates()*** in the module below.

Python code:
database_age_plot.py



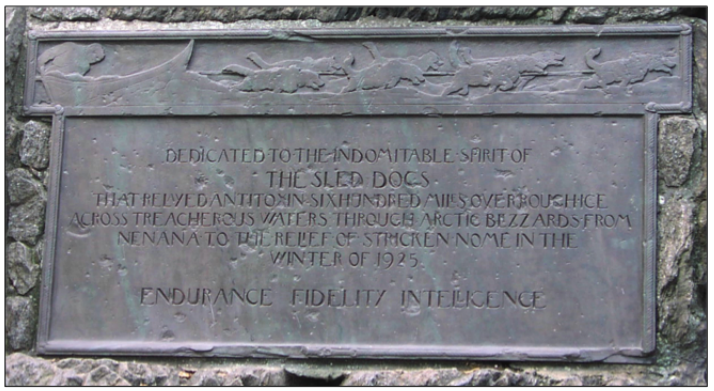

```
Number of dates: 25,757
Date range:
2007-10-08  to  2018-08-09
Median date: 2014-11-09
```

Query Results -- Part II

- Most of the data was collected from the years 2013 to 2018 -- the data is out-of-date. The oldest data is from year 2007.
  The median date is November, 2014. This means half the data is more than 4 years old.
- The surprise find was *inscription* data is listed, as shown by the Balto statue. I walked over there and it's accurate! That's interesting and shows the potential of the Open Street Maps project.

The Inscription at Balto Statue

Central Park, New York



Conclusions

- Much data is missing. For example, for cuisine listed as Burger, on the Upper West Side (in zip codes 10023, 10024, or 10025) only ten places are listed. There are ***many*** more than ten!
- Data accuracy is good, not excellent. For example, all RadioShack stores in New York City are permanently closed, but some are in the dataset. Additionally, the Jackson Hole burger restaurant listed is closed.

Database advisory: When retrieving data, use the ***timestamp*** field in the *Nodes* and *Ways* tables to determine the date when the data was entered. In large cities, and especially in New York, stores and restaurants are constantly changing. Information older than one year should be double-checked on the web.

- *Go Map!!* is an iOS app to create and edit information in Open Street Map.
  Open Street Map should advertise the app on their website home page to encourage their audience to contribute.

Data wrangling is an arduous, time-consuming endeavor. The approach taken here is to programmatically clean the data before loading it into the database. Then, queries are run against the database to analyze the data.

References

1. Open Street Map, https://www.openstreetmap.org (https://www.openstreetmap.org)
2. Open Street Map Wiki, https://wiki.openstreetmap.org/wiki/Main_Page (https://wiki.openstreetmap.org/wiki/Main_Page)
3. Python Software Foundation, Python Documentation, https://docs.python.org (https://docs.python.org)
4. SQLite Database, https://www.sqlite.org (https://www.sqlite.org)
5. Wikipedia, https://en.wikipedia.org/wiki/Data_wrangling (https://en.wikipedia.org/wiki/Data_wrangling)
6. Udacity Inc., notes from lectures and coding exercises
7. Python Software Foundation, PEP 257 -- Docstring Conventions, https://www.python.org/dev/peps/pep-0257/ (https://www.python.org/dev/peps/pep-0257/)