# OpenStreetMap Project

OpenStreetMap Project 1

Introduction 2

Data Wrangling 2

Data Overview 4

Data Exploration 5

Improvements 7

Data Quality 7

Data Analysis 7

Conclusion 7

## Introduction

* **City/district**: Brooklyn New York
  + <https://www.openstreetmap.org/node/158857828>
  + **Brooklyn** is the most populous of [New York City](https://en.wikipedia.org/wiki/New_York_City)'s five [boroughs](https://en.wikipedia.org/wiki/Borough_(New_York_City)), with Census-estimated 2,636,735 residents in 2015. [[1]](#footnote-1)
  + I have no special reason for choosing Brooklyn. It might be I have seen it many times in movies and I wanted to know more about this borough statistically.
  + OSM download address: <https://s3.amazonaws.com/metro-extracts.mapzen.com/brooklyn_new-york.osm.bz2>
* **Database**: MongoDB
* **Note**: Only Python was used for data wrangling and exploration.

## Data Wrangling

1. Use the functions provided in the quiz to list the streets that do not meet the specifications.
   1. Sometime full street type is used whilst sometime short forms are used such as Ave. St.
   2. These short forms can come in any case, for example, 9th ST., South 4th St.

**Solution**: Regular expressions were used to normalize the street types.

1. Postcode: most postcodes in the Brooklyn dataset are in valid 5-digit format with a few exceptions:
   * 5+4 format.
   * More than one code such as *11221;11222*.
   * Invalid format such as *NY 11211*
   * No postcode found: *(718) 778-0140*.

**Query**: c.distinct("addr.postcode",{"addr.postcode":{"$exists":1}})

[u'10001', u'11201', u'07030', u'07302', u'11379', u'11219', u'11232', u'11416', u'11418', u'11238', u'11222', u'11211', u'11234', u'11249', u'11235', u'11203', u'11216', u'11209', u'10038', u'10014', u'11224', u'10002', u'10003', u'11378', u'10011', u'10013', u'11385', u'11231', u'11201-2483', u'11215', u'10009', u'10007', u'11206', u'11224-4003', u'11217', u'11377', u'07030-5774', u'11223', u'10012', u'11233', u'10282', u'NY 11201', u'11205', u'11218', u'11226', u'11373', u'10005', u'11204', u'10010', u'11230', u'11220', u'10011-6832', u'11221', u'11374', u'11208', u'11214', u'11375', u'11210', u'11236', u'11213', u'11207', u'11239', u'11417', u'11414', u'11421', u'11415', u'11225', u'11229', u'11228', u'NY 10002', u'11212', u'NY 11221', u'11237', u'11368', u'11367', u'10004', u'10012-3332', u'10006', u'11101', u'10280', u'10281', u'10275', u'11104', u'11419', u'11694', u'11697', u'07310', u'07311', u'(718) 778-0140', u'10018', u'11420', u'10048', u'11215-9993', u'11201;11231', u'11231;11230', u'11214;11223', u'11232-2400', u'11251', u'10023', u'10002-1013', u'10016']

**Solution**: finds all the documents where *addr.postcode* exists and update the records if necessary:

* (ZIP + 4) format: No need to change since it is valid.
* Multiple codes or codes in invalid format: extract all postcodes and change the data type of postcode field to list.
  + NY 11211 🡪 [11211]
  + 11221;11222 🡪[11221, 11222]
* No valid postcode is found: set the postcode to *None*.
  + (718) 778-0140 🡪 None

Here are the postcodes after cleanup.

unique postcodes [u'10001', u'11201', u'07030', u'07302', u'11379', u'11219', u'11232', u'11416', u'11418', u'11238', u'11211', u'11222', u'11234', u'11249', u'11235', u'11203', u'11216', u'11209', u'10038', u'10014', u'11224', u'10002', u'11378', u'10003', u'10011', u'10013', u'11385', u'11231', u'11201-2483', u'11215', u'10009', u'10007', u'11206', u'11224-4003', u'11217', u'11377', u'07030-5774', u'11223', u'10012', u'11233', u'10282', u'11205', u'11226', u'11218', u'11373', u'10005', u'11204', u'10010', u'11230', u'11220', u'10011-6832', u'11221', u'11374', u'11208', u'11214', u'11375', u'11210', u'11236', u'11213', u'11207', u'11239', u'11417', u'11414', u'11421', u'11415', u'11225', u'11229', u'11228', u'11212', u'11237', u'11368', u'11367', u'10004', u'10012-3332', u'10006', u'11101', u'10280', u'10281', u'10275', u'11104', u'11419', u'11694', u'11697', u'07310', u'07311', None, u'10018', u'11420', u'10048', u'11215-9993', u'11232-2400', u'11251', u'10023', u'10002-1013', u'10016'

1. There are a lot of compound keys such as *addr:street* or *xmas:location*.

**Solution**: Instead of having two separate tags, store related tags in a dictionary in a hierarchical way.

{“addr”:

{“street”:”XYZ”}

}

{“building”:

{

“building”:”yes”,

“colour”:”red”

}

}

1. Heights are stored as strings and some values are numbers only and some have units such as *10 ft* or *10 feet*.

**Solution**: use a regular expression to convert them to floating point numbers.

1. Timestamps are in string format

**Solution**: use a regular expression to converted from strings to JSON datatime format.

## Data Overview

* OSM file size: 663.2MB
* JSON file size: 730.3MB
* print "documents:", c.find().count()

documents: 2967672

* print "nodes:",c.find({"type":"node"}).count()

nodes: 2478629

* print "ways:",c.find({"type":"way"}).count()

ways: 487280

* print "relations:",c.find({"type":"relation"}).count()

relations: 6

* print "unique users:", len(c.distinct("created.user"))

unique users: 1297

* print "number of cafes:", c.find({"amenity":"cafe"}).count()

number of cafes: 325

* print "number of schools:", len(get\_db().ny.distinct("name",{"amenity":"school"}))

number of schools: 544

* print "number of parks:", c.find({"leisure":"park"}).count()

number of parks: 780

* print "number of buildings:", c.find({"building":"yes"}).count()

number of buildings: 393537

* print "facbook:", c.find({"contact.facebook":{"$exists":1}}).count()

print "twitter:", c.find({"contact.twitter":{"$exists":1}}).count()

print "instagram:", c.find({"contact.instagram":{"$exists":1}}).count()

print "google\_plus:", c.find({"contact.google\_plus":{"$exists":1}}).count()

print "yelp:", c.find({"contact.yelp":{"$exists":1}}).count()

facbook: 7

twitter: 4

instagram: 1

google\_plus: 6

yelp: 3

## Data Exploration

* Biggest street (with most addresses)

pipeline = [

{"$match":{"addr.street":{"$exists":1}}},

{"$group": {"\_id": "$addr.street", "count": {"$sum": 1}}},

{"$sort":{"count":-1}},

{"$limit":1}

]

Streets with most addresses:

{u'\_id': u'Bedford Avenue', u'count': 2062}

* Street with the most parking lots:

pipeline = [

{"$match":{"amenity":"parking"}},

{"$match":{"addr.street":{"$exists":1}}},

{"$group": {"\_id": "$addr.street", "count": {"$sum": 1}}},

{"$sort":{"count":-1}},

{"$limit":1}

]

Streets with most parking lots:

{u'\_id': u'West 5th Street', u'count': 4}

* The tallest building

pipeline = [

{"$match":{"height":{"$exists":1}}},

{"$match":{"building":"yes"}},

{"$sort":{"height":-1}},

{"$limit":1}

]

{u'\_id': ObjectId('574318a07f8cb505e1659e94'), u'building': u'yes', u'building:colour': u'black', u'building:levels': u'54',

u'created': {u'changeset': u'22026081', u'timestamp': datetime.datetime(2014, 4, 29, 16, 16, 31), u'uid': u'1781294', u'user': u'Rub21\_nycbuildings', u'version': u'1'}, u'height': 226.0, u'id': u'278039446', u'name': u'One Liberty Plaza', u'node\_refs': [u'2824697726', u'2824697766', u'2824697920', u'2824697781', u'2824697731', u'2824697726'], u'nycdoitt:bin': u'1001068', u'roof:material': u'concrete', u'roof:shape': u'flat', u'start\_date': u'1973', u'type': u'way', u'wikipedia': u'en:One Liberty Plaza'}

\* As discussed later, this result might not be correct because different units are used.

* Top 3 sports:

pipeline = [

{"$match":{"sport":{"$exists":1}}},

{"$group": {"\_id": "$sport", "count": {"$sum": 1}}},

{"$sort":{"count":-1}},

{"$limit":3}

]

Top 3 sports:

{u'\_id': u'tennis', u'count': 202} {u'\_id': u'basketball', u'count': 188}

{u'\_id': u'baseball', u'count': 181}

## Improvements

### Data Quality

The use of tags gives users the ability and flexibilty to add extra information to the map. However, it can also cause a lot of problems for data analysis.

* In total, in the database I used, there are 922 unique tag keys and most of them are not well documented. Data analysts have to go through the whole list, guess the meaning of each tag. Sometimes, they can be misled. For example, *park* is one of the tags, if you use it to find out how many parks in Brooklyn, you will be disappointed.

print "number of parks (using park tag):", c.find({"park":{"$exists":1}}).count()

* There are also similar tag keys. For example,
  + 'leisure',
  + 'leisure\_1',
  + 'leisure\_2',

This is because some places have multiple leisure functions. The data are not a big issue. The problem is that different ways are used to handle the scenarios where there are multiple values for one key.

* + Postcode: use commas to separate multiple codes
  + Leisure: add more keys

A standard way should be defined to handle this case. I think the one with a single key and comma-separated values is better than the other since it is more flexible and fewer keys are created.

* Inconsistent units. In *height* fields, some of them have units like *feet*, *ft* and some of them are just numbers. Even worse, different units are used. For example, the height of One Liberty Plaza uses meter as the unit. Inconsistent units make it impossible to conduct reliable data analysis.

Here are my suggestions to improve the data quality:

* There should be a list of pre-defined tags and each tag should clearly define its name, description, possible values, valid format and data type.
* New tags should be unambiguously defined as well (name, description, possible values, valid format, data type) and pre-approved.
* If a new tag is suspiciously similar to an existing tag, the users should see the hint.
* Validation checks should be performed before the data are entered into the system.

The potential problems with my suggestions are users may not be patient enough to wait for the approval and there might be short of human resources to check and approve new tags.

### Data Analysis

* Check the quality of data and rate the accuracy and reliability. This can be done by choosing samples and cross-referencing. It can also be done by listing all unique values in the data set.
* Exclude inaccurate/incomplete data from the data analysis.
* In the final analysis report, data quality should be examined and the reliability of the analysis and potential risks of using the analysis should be evaluated.

The potential problems with my suggestions are sometimes it is very hard to do cross-referencing, as data are not always available. And the samples chosen are also an important factor of the quality of data auditing.

### Conclusion

In an era of big data, we often have to deal with large amount, semi-structured data whose quality may vary. Some data might be incomplete, inaccurate and even misleading. Data collectors should try to normalize and validate their data and data analysts should always be aware of the importance of data auditing and wrangling.

1. https://en.wikipedia.org/wiki/Brooklyn [↑](#footnote-ref-1)