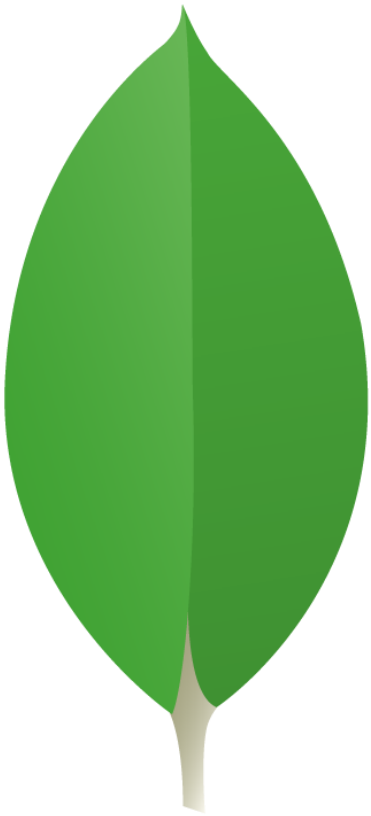
Udacity Data Analyst Nanodegree



Project 3:

Data Wrangling with MongoDB

Daniel Mercier

1 Problems Encountered in Your Map

For my project, I used the dataset for the Ottawa-Gatineau region as defined at Mapzen. Situated in Canada, Ottawa-Gatineau straddles both the English-speaking province of Ontario and French-speaking province of Quebec. The dataset presents the unique challenge of working with entries in two languages.

I'll first describe the general data cleaning problems I encountered, followed by the language-related ones.

1.1 Standardizing addr:street values

The values for `addr:street` were standardized by:

- Expanding street type abbreviations.
- Expanding cardinal directions.
- Moving cardinal directions to the front of the street name.
- Removing all excess whitespace.
- Capitalizing words starting with a lowercase letter.

I used regular expressions to detect these problems. Regexes for each operation were contained in a list of lists, each of which contained a regex along with a correction for any matches.

For example, here is the regex/correction list for street types of type 'Street':

```
[re.compile(r'\sSt.\s|\sSt\.?$', re.IGNORECASE), 'Street']
```

Here is the result for an element with all the above cleaning operations applied:

- 'Baker St. south' -> 'South Baker Street'

The full list of regexes used in the cleaning process can be viewed in Appendix 1, while the assertion tests used for verification can be viewed in Appendix 2.

1.2 Standardizing addr:city values

As Ottawa and Gatineau are separate cities and the dataset also includes many surrounding municipalities, I couldn't standardize all entries to a single city name.

However, Ottawa is an amalgamated city, and some entries still have their pre-amalgamation municipality names in the `addr:city` value. Additionally many of the 40 or so unique regions are also prefixed by their municipality type (e.g.: 'City of Ottawa' instead of 'Ottawa'). As a result, the approach I took was to standardize `addr:city` values to show only their name.

For example:

- City of Ottawa -> Ottawa
- Town of Carleton Place -> Carleton Place
- Village of Casselman -> Casselman

This approach does take away extra information from the `addr:city` field. However, as municipality type and municipality name are distinct pieces of information, they would probably be better represented by breaking them out into separate fields if one were interested in both of them.

1.3 Bilingual street types

In French, the street type is given at the beginning of an address rather than at the end. This means I needed to distinguish French entries from English; if I only checked for street type at the end of a street name, every French street name would be treated as a unique English street type.

This required filtering entries by iteratively building expected street type lists for each language. I started by auditing English street types in descending order of frequency, picking out the most common ones and adding them to our English expected list.

I then used this list as an English entry filter when auditing French street types, building a French expected list in the same way as the English one.

Finally, I used the combined English and French expected list as a filter to show entries with irregularities or valid street types with low counts that I may have initially missed.

1.4 Abbreviation overlap

The dataset presents us with another kind of challenge in the cleaning phase of the project. While the actual spelling of most French and English street types differ, some of their shared abbreviations do not.

For example:

Table 1: Abbreviations and their English and French expansions

ABBREVIATION	EN. EXPANSION	FR. EXPANSION
S.	South	Sud
N.	East	Nord
E.	East	Est
Cr.	Crescent	Croissant

The table above shows how significant this issue this can be – three of the examples are commonly occurring cardinal directions.

Distinguishing which language an abbreviation belongs to during the cleaning process would likely introduce errors and processing overhead, meaning we effectively must choose only one language to expand an abbreviation to.

1.5 Unicode characters

As French uses accented characters, everything needs to be processed in Unicode. I used Python 3, as it handles Unicode strings in most operations without any extra work in comparison to Python 2. However, Unicode is still not fully supported in all of the core Python modules

For example, Python's logging module, which I used to output auditing, testing and query results to an external file, would throw exceptions when encountering a Unicode string. This was not a serious issue, as console output worked just fine, but it did require me to disable logging for some MongoDB query functions.

2 Overview of the Data

This section gives an overview of the Ottawa-Gatineau dataset. Queries and their abbreviated results are shown alongside each other.

2.1 Dataset Characteristics

Database structure

Table 2: Dataset file sizes

FILENAME	SIZE
`ottawa_canada.osm`	407 MB (417, 067 KB)
`ottawa_canada.osm.json`	578 MB (592, 763 KB)

Table 3: Queries for counts of documents, nodes and ways

TYPE	QUERY	COUNT
documents	`db.ottawa.find().count())`	1964211
nodes	`db.ottawa.find({'type' : node'}).count())`	1770406
ways	`db.ottawa.find({'type' : 'way'}).count())`	193687

Data sources

Displays the data sources used by contributors and their usage count in descending order.

Table 4: Data sources query with top 10 results

QUERY

```
{'$group' :
    {
    '_id' : '$source',
    'count' :  {'$sum' : 1}
    }
},
{'$match' : {'_id' : {'$ne' :
None}}},
{'$sort' : {'count' : -1}}
```

RESULTS

```
[{'_id': 'CanVec 6.0 - NRCan', 'count':
148248},
 {'_id': 'NRCan-CanVec-7.0', 'count': 62445},
 {'_id': 'http://data.ottawa.ca/dataset/',
'count': 40911},
 {'_id': 'NRCan-CanVec-10.0', 'count': 34184}
 {'_id': 'Bing', 'count': 25036},
 {'_id': 'CanVec 6.0 - NRCan; NRCan-CanVec-
7.0', 'count': 15283},
 {'_id': 'Geobase_Import_2009', 'count':
3295},
 {'_id': 'survey (KofM)', 'count': 2032},
 {'_id': 'City of Ottawa', 'count': 1968},
 {'_id': 'GeobaseNHN_Import_2009', 'count':
915},
  ...]
```

The majority of the data has been sourced from versions of Natural Resources Canada's CanVec datasets. A surprising amount is also sourced from Microsoft's Bing search engine.

2.2 User Statistics

Top contributors

Displays users and their contribution count in descending order.

Table 5: Top contributors query with top 10 results

QUERY

```
{'$group' :
    {
    '_id' : '$created.user',
    'count' :  {'$sum' : 1}
    }
},

{'$sort' : {'count' : -1}}
```

RESULTS

```
[{'_id': 'andrewpmk', 'count': 531635},
 {'_id': 'Johnwhelan', 'count': 435725},
 {'_id': 'LogicalViolinist', 'count': 231268}
 {'_id': 'andrewpmk_imports', 'count':
168540},
 {'_id': 'mcwetboy', 'count': 136036},
 {'_id': 'Rps333', 'count': 82537},
 {'_id': 'Keeper of Maps', 'count': 44207},
 {'_id': 'DenisCarriere', 'count': 28873},
 {'_id': 'KarenB', 'count': 19005},
 {'_id': 'dommage', 'count': 17809}
  ...]
```

Based on the results of the query we can see that a significant amount of the entries come from a relatively small group of users.

Users with a single entry

Returns a list of users who have made a single entry.

Table 6: Query and count for users with single entry

QUERY

```
{'$group':
    {
     '_id' : '$created.user',
     'count' : {'$sum': 1}
    }
},
{'$group':
     {
     '_id' : '$count',
     'users' : {'$sum': 1}
     }
},
{'$sort' : {'_id' : 1}},
{'$limit': 1 }
```

RESULTS

```
[{'_id': 1, 'users': 139}]
```

Number of unique users

Returns a total count of all unique users.

Table 7: Query and count for all unique users

QUERY	RESULTS
`len(db.ottawa.distinct('created.user'))`	863

List of unique users

Returns a list of all unique users.

Table 8: Unique users query with first 10 results

QUERY

```
db.ottawa.distinct('created.user')
```

RESULTS

```
['andrewpmk',
 'Sensei Marc',
 'OC_Transpo',
 'mcwetboy',
 'Jscott',
 'dommage',
 'Terry Barber',
 'jon_snow',
 'Émile Cardinal',
 'Andre613',
  ...]
```

2.3 Geographical Characteristics

City by regions

Displays city regions and their frequency in descending order.

Table 9: City region by frequency query with top 10 results

QUERY

```
{'$group' :
    {
    '_id' : '$address.city',
    'count' :  {'$sum' : 1}
    }
},
{'$sort' : {'count' : -1}}
```

RESULTS

```
[{'_id': None, 'count': 1691171},
 {'_id': 'Ottawa', 'count': 124336},
 {'_id': 'Gatineau', 'count': 115318},
 {'_id': 'North Dundas', 'count': 3416},
 {'_id': 'Mississippi Mills', 'count': 3373},
 {'_id': 'North Grenville', 'count': 3119},
 {'_id': 'Clarence-Rockland', 'count': 2770},
 {'_id': 'Russell', 'count': 2106},
 {'_id': 'Chelsea', 'count': 1832},
 {'_id': 'Carleton Place', 'count': 1715},
  ...]
```

Cuisine Types

Displays the city's cuisine types and their popularity in descending order.

Table 10: Cuisine type by frequency query with top 10 results

QUERY

```
{'$group' :
    {
    '_id' : '$cuisine',
    'count' :  {'$sum' : 1}
    }
},

{'$sort' : {'count' : -1}}
```

RESULTS

```
[{'_id': None, 'count': 1963511},
 {'_id': 'coffee_shop', 'count': 133},
 {'_id': 'pizza', 'count': 81},
 {'_id': 'burger', 'count': 80},
 {'_id': 'sandwich', 'count': 45},
 {'_id': 'italian', 'count': 35},
 {'_id': 'chinese', 'count': 34},
 {'_id': 'regional', 'count': 28},
 {'_id': 'american', 'count': 22},
 {'_id': 'asian', 'count': 18},
  ...]
```

3 Other Ideas About the Datasets

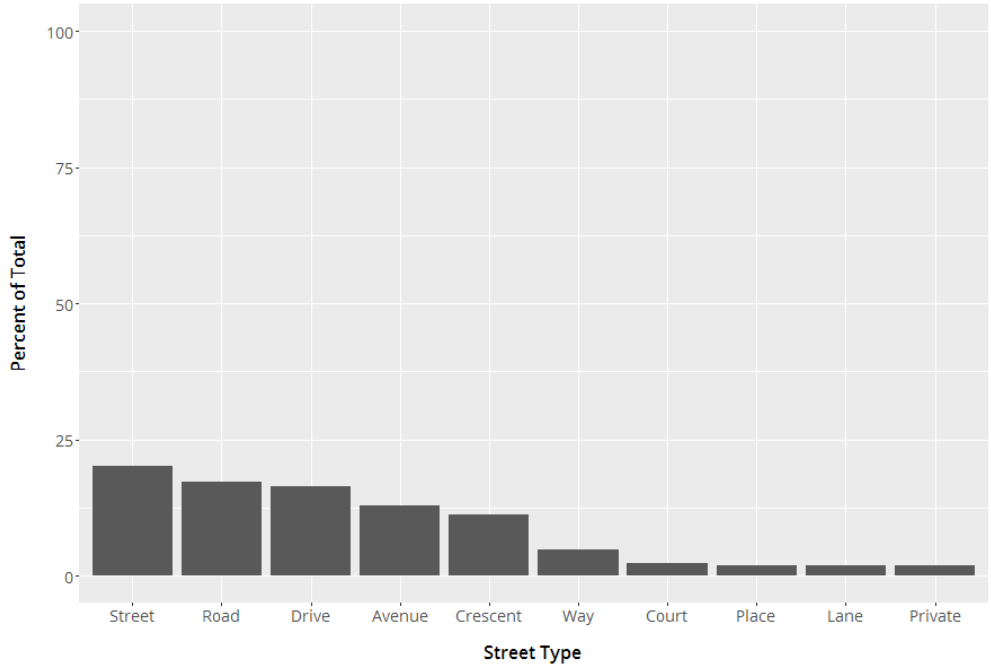
3.1 Dataset Analysis

Working with a bilingual dataset may make for a more demanding data munging experience, but it also presents an opportunity to do some exploratory analysis using the French and English data.

As a starting point, I explored whether any differences in distribution exist between the street types for each language, plotting the top 10 most frequently occurring ones for each in R.

First, the distribution of the English data:

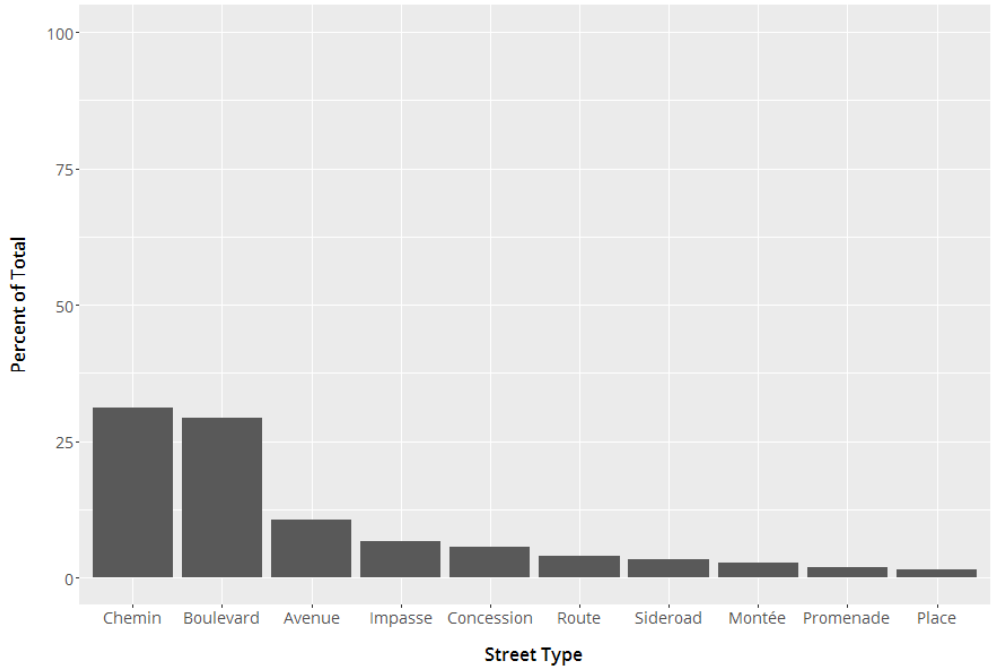
Figure 1: Distribution of top 10 English street types



As we might expect, Street and Road are the two most frequently occurring types, but not by much – Drive, Avenue and Crescent also occur fairly regularly.

Next, the distribution of the French data:

Figure 2: Distribution of top 10 French street types



Right away, we can see that the distribution is much more concentrated than in the English data.

Table 11: English and French street types by percent of total count

TYPE	COUNT	PERCENT
Street	30611	20.1
Road	26345	17.3
Drive	24950	16.4
Avenue	19571	12.9
Crescent	17124	11.3
Way	7100	4.7
Court	3292	2.2
Place	2933	1.9
Lane	2876	1.9
Private	2864	1.9

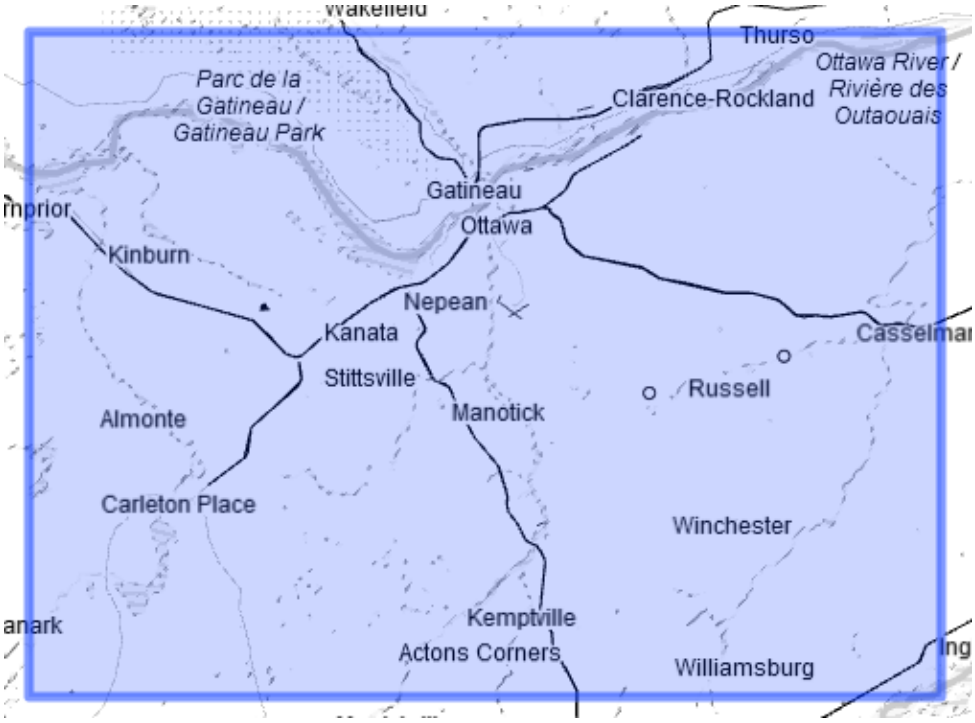
TYPE	COUNT	PERCENT
Chemin	8190	31.2
Boulevard	7673	29.2
Avenue	2817	10.7
Impasse	1746	6.6
Concession	1489	5.7
Route	1015	3.9
Sideroad	882	3.4
Montée	730	2.8
Promenade	463	1.8
Place	364	1.4

Taking a look at the actual numbers, we can see that while the top 2 street types in the English data make up 37.4% of the total, in the French data they take a much larger 60.4% share:

So far I've talked about the division in the data in terms of language, but my time as a resident of Ottawa leads me to believe that this isn't the whole story.

To get a better idea of why this might be, let's take a closer look at the area of Ottawa-Gatineau that the dataset covers:

Figure 3: Area covered by Ottawa-Gatineau dataset



That grey line snaking along the top half of the map is the Ottawa River. It divides Ottawa and Gatineau along geographic and provincial lines, and more softly along linguistic and cultural ones. Within the confines of the dataset, the Ottawa region covers a much larger area, encompassing urban, suburban and country terrain. This might explain why street types associated with suburban areas such as Way, Crescent and Avenue appear more frequently.

Gatineau, however, is heavily defined by Gatineau Park, a massive conservation park that covers 361 square kilometres. Situated alongside the urban core, its influence, combined with the more limited street types usually present in urban settings, could explain the difference we see in the street type distributions.

Although this is not a definitive answer, I believe that exploring similar questions using language and geography could yield insights that help businesses and individuals with logistical and operational decisions.

For example:

- Based on the prevalence of cuisine types in each region, what kind of response could one expect to a new restaurant opening in Gatineau compared to Ottawa?
- How accessible would a new office be given the types of streets surrounding it?

Additionally, the vast majority of entries in the dataset contain GPS coordinates, and given MongoDB's ability to use geospatial indexing, the dataset could be used to answer questions such as these at a fine level of detail.

One caveat is the relatively old age of the data. Most of it is sourced from old versions of the CanVec dataset – CanVec 10.0, the latest version used in the dataset, was released in February 2012, while CanVec 6.0, from which the majority of the data was sourced, was released in April 2010. For any kind of time-sensitive analysis, the data here would be of questionable value.

3.2 Recommendations

I believe the following recommendations would improve the dataset's quality and accessibility considerably.

Update primary data sources

As mentioned previously, the majority of the data is sourced from old versions of CanVec datasets. This data has recently been transitioned into a new format called CanVec+. The Canadian Government offers free access to CanVec+ and updates it bimonthly. However, the data is offered in the GML (Geography Markup Language) format. GML is an extension of XML, and has a similar format to OSM data. As such, it should be possible to convert it to OSM, or to use the cleaning and shaping process in this project to format the GML data into a JSON file that mirrors the fields found in an OSM file.

Separate English and French data

Processing a dataset with two languages simultaneously adds a lot of unnecessary complexity to the all aspects of the data wrangling process.

Based on my experience with this project, I would recommend that the dataset be separated by language into two unique datasets. Each would benefit from a language-specific data wrangling process free of linguistic compromises and trade-offs.

In this particular instance, this would mean discarding the predefined dataset from Mapzen and manually creating separate datasets for both the Ottawa and Gatineau region. Defining the regions might require more upfront work, but would likely result in greater accuracy and ease of processing.

4 SOURCES

1. **Unicode HOWTO.** Python 2 and 3 Unicode support. *Python Docs.* [Online] https://docs.python.org/3/howto/unicode.html.
2. **RegexOne.** Regular expressions tutorial. [Online] http://regexone.com.
3. **Open Street Map Wiki.** Canada:Ontario:Ottawa. [Online] http://wiki.openstreetmap.org/wiki/Canada:Ontario:Ottawa.
4. **ElementTree XML API.** Documenation on ElementTree. *Python Docs.* [Online] https://docs.python.org/3/library/xml.etree.elementtree.html.
5. **Al Sweigart.** Automate the Boring Stuff with Python. no starch press, 2015. ISBN-10: 1-59327-599-4, ISBN-13: 978-1-59327-599-0
6. **Mark Lutz.** Python Pocket Reference, 4th edition. Sebastopol: O'Reilly Media, Inc., 2010. ISBN: 978-0-596-15808-8.

5 Link to Map Position

5.1 Links

Link to metro extract (search for Ottawa, Canada):

https://mapzen.com/data/metro-extracts

Link to dataset:

https://s3.amazonaws.com/metro-extracts.mapzen.com/ottawa_canada.osm.bz2

5.2 Reason for selection

Being a long-term resident, I selected the Ottawa-Gatineau region to look at a city I feel I know from a new perspective. The bilingual nature of the dataset also offered a unique data wrangling challenge.

6 APPENDIX 1 – CLEANING.PY REGEXES

```
### Regex matching patterns

type_regexes = [
    [re.compile(r'\sSt.\s|\sSt\.?$', re.IGNORECASE), 'Street'],
    [re.compile(r'\s?Rd\.?\s|\sRd\.?$', re.IGNORECASE), 'Road'],
    [re.compile(r'\s?Dr\.?\s|\sDr\.?$', re.IGNORECASE), 'Drive'],
    [re.compile(r'\s?Ave\.?\s|\sAve\.?$', re.IGNORECASE), 'Avenue'],
    [re.compile(r'\s?Cr\.?\s|\sCr\.?$', re.IGNORECASE), 'Crescent'],
    [re.compile(r'\s?Wy\.?\s|\sWy\.?$', re.IGNORECASE), 'Way'],
]

type_regexes_french = [
    [re.compile(r'\sR.\s|\sR\.?$', re.IGNORECASE), 'Rue'],
    [re.compile(r'\s?Ch\.?\s|\sCh\.?$', re.IGNORECASE), 'Chemin'],
    [re.compile(r'\s?Blvd\.?\s|\Blvd\.?$', re.IGNORECASE), 'Boulevard'],
    [re.compile(r'\s?Ave\.?\s|\sAve\.?$', re.IGNORECASE), 'Avenue'],
    [re.compile(r'\sImp.\s|\sImp\.?$', re.IGNORECASE), 'Impasse'],
    [re.compile(r'\s?Conc\.?\s|\sConc\.?$', re.IGNORECASE), 'Concession'],
]

cardinal_regexes = [
    [re.compile(r'\s?(?<!\S)N\.?\s|\s(?<!\S)N\.?$', re.IGNORECASE), 'North'],
    [re.compile(r'\s?(?<!\S)W\.?\s|\s(?<!\S)W\.?$', re.IGNORECASE), 'West'],
    [re.compile(r'\s?(?<!\S)E\.?\s|\s(?<!\S)E\.?$', re.IGNORECASE), 'East'],
    [re.compile(r'\s?(?<!\S)S\.?\s|\s(?<!\S)S\.?$', re.IGNORECASE), 'South']
]

move_cardinal_regexes = {
    'North': re.compile(r'(North[^\w]|North$)', re.IGNORECASE),
    'West': re.compile(r'(West[^\w]|West$)', re.IGNORECASE),
    'East': re.compile(r'(East[^\w]|East$)', re.IGNORECASE),
    'South': re.compile(r'(South$[^\w]|South$)', re.IGNORECASE)
}

city_regexes = [
    [re.compile(r'.*of\s|.*Of\s'), '']
]

problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')
```

7 APPENDIX 2 – CLEANING.PY ASSERTION TESTS

```
def test_functions():
    # Test process_map function
    data = process_map('..\\[Data]\\ottawa_canada_sample_tiny.osm', True)
    correct_first_elem = {
        "name": "Shell",
        "id": "969421551",
        "created": {
            "user": "Johnwhelan",
            "uid": "186592",
            "version": "1",
            "timestamp": "2010-10-30T00:03:00Z",
            "changeset": "6223495"
        },
        "address": {
            "housenumber": "19",
            "city": "Ottawa",
            "street": "O'hara Drive"
        },
        "website": "http://www.shell.ca/",
        "source": "CanVec 6.0 - NRCan",
        "amenity": "fuel",
        "type": "node",
        "pos": [
            45.3839697,
            -75.9596713
        ]
    }

    assert data[0] == correct_first_elem

    # Test check_for_extended_addr function
    assert check_for_extended_addr('addr::') == True
    assert check_for_extended_addr('addr:') == False

    # Test update_city_name function
    assert update_city_name('City of Ottawa') == 'Ottawa'
    assert update_city_name('Township Of Tay Valley') == 'Tay Valley'

    # Test update cardinal direction function
    assert update_cardinal_direction('Baker Street S.') == 'South Baker Street'
    assert update_cardinal_direction('Baker Street E.') == 'East Baker Street'
    assert update_cardinal_direction('Baker Street N') == 'North Baker Street'
    assert update_cardinal_direction('Baker Street W') == 'West Baker Street'

    # Test move cardinal direction function
    assert move_cardinal_direction('Baker Street South',
                                   'South') == 'South Baker Street'
    assert move_cardinal_direction('Baker Street East',
                                   'East') == 'East Baker Street'
    assert move_cardinal_direction('Baker Street North',
                                   'North') == 'North Baker Street'
    assert move_cardinal_direction('Baker Street West',
                                   'West') == 'West Baker Street'

    # Test remove_whitespace function
    assert remove_whitespace('Baker  Street') == 'Baker Street'
    assert remove_whitespace('Baker   Street') == 'Baker Street'
    assert remove_whitespace(' Baker   Street ') == 'Baker Street'

    # Test update_street_type function
    assert update_street_type('South Ash St.') == 'South Ash Street'
    assert update_street_type('St. Ash St.') == 'St. Ash Street'
    assert update_street_type('South Ash Rd.') == 'South Ash Road'
    assert update_street_type('South Ash Dr.') == 'South Ash Drive'
```

```
assert update_street_type('South Ash Ave.') == 'South Ash Avenue'
assert update_street_type('South Ash Cr.') == 'South Ash Crescent'
assert update_street_type('South Ash Wy') == 'South Ash Way'

# Test capitalize_tag function
assert capitalize_tag('baker Street') == 'Baker Street'
assert capitalize_tag('south baker street') == 'South Baker Street'

# Test update_tag function function
assert update_tag('addr:city', 'City of Ottawa') == 'Ottawa'
assert update_tag('addr:street', 'Ash St. S.') == 'South Ash Street'

# Test clean_tag function
assert clean_tag('addr:street', 'Chanonhouse st.') == 'Chanonhouse Street'
assert clean_tag('addr:street', 'St. Jerome St') == 'St. Jerome Street'
assert clean_tag('addr:street', 'South Ash Cr ') == 'South Ash Crescent'
assert clean_tag('addr:street',
                 'Baker   Street south') == 'South Baker Street'
```