Data Wrangle OpenstreetMaps Data

Student Notes Code Review 3 Project Review Download submission

Lasso.py 3

```python
'''
    The lasso module contains all the functions used in data wrangling the Open Street Map Da
'''

import xml.etree.cElementTree as ET
import re
import codecs
import json
from collections import defaultdict


#  Regex
```

AWESOME

Outstanding commenting etiquette!

```python
RE_PROBLEM_CHARS = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')
RE_POSTAL_CODE = re.compile(r"^([a-zA-Z]\d[a-zA-Z] ?\d[a-zA-Z]\d)$")
# http://stackoverflow.com/questions/16614648/canadian-postal-code-regex


# Default Dicts used in audit functions
```

AWESOME

You have avoided in-line comments which often extend the character-per-line preference of 80. Wel done!

```python
def def_dict_2():
    '''
    defaultdict two dict deep with default of set
    '''
    return defaultdict(lambda: defaultdict(set))


def def_dict_3():
    '''
    defaultdict three dict deep with default of set
    '''
    return defaultdict(lambda: defaultdict(lambda: defaultdict(set)))


#########################
### Auditing the data ###
#########################
```

```python


def dictify_element_and_children(element, atr_d=def_dict_2(), st_atr_d=def_dict_3(), s_st_d=d
```

AWESOME

Object naming corresponds to the objects intended function. Well done!

```python
    '''
    From each element in the xml tree creats/adds summary dictionaries to better understand t

    return:
        1 - attrib_dict (atr_d): should return all potential attributes with a set of all ans
        2 - sub_tag_attrib_dict (st_atr_d): return sub_tag: sub_tag.attrib.keys()
        3 - sub_subtag_dict (s_st_d): return sub_tag: sub_tag.children
        4 - tag_k_v_dict: for tag sub_tag:
    '''
    for key, val in element.attrib.items():
        atr_d[element.tag][key].add(val)
    for sub_tag in element.iter():
        child_set = {el.tag for el in list(sub_tag)}
        if child_set != set():
            s_st_d[element.tag][sub_tag.tag].update(child_set)
        for key, val in sub_tag.attrib.items():
            st_atr_d[element.tag][sub_tag.tag][key].add(val)
        if sub_tag.tag == 'tag':
            tag_k_v_dict[element.tag][sub_tag.attrib['k']].add(sub_tag.attrib['v'])

    return atr_d, st_atr_d, s_st_d, tag_k_v_dict


def summarizes_data_2_tags_deep(filename):
    '''
    uses dictify_element_and_children to loop through entire xml tree creating summary data.
    '''
    atr_d = def_dict_2()
    st_atr_d = def_dict_3()
    s_st_d = def_dict_2()
    tag_k_v_dict = def_dict_2()

    for _, element in ET.iterparse(filename):
        dictify_element_and_children(element, atr_d, st_atr_d, s_st_d, tag_k_v_dict)
    return atr_d, st_atr_d, s_st_d, tag_k_v_dict


def check_keys_list(dict_key_list):
    '''
        checks a list of dictionary keys for problem characters
    '''
    problem_keys = []
    for key in dict_key_list:
        if RE_PROBLEM_CHARS.search(key):
            problem_keys.append(key)
    return problem_keys


def process_audit_address_type(tag_k_v_dict, directions=()):
    '''
    loops though all street addesses putting all the street types in a set
    if the last word in the steet addess is a direction (E, W, N, S)
```

```
        it uses the second last word in the street address
    if not
        it uses the last word in the street address
    '''
    street_types = set()
    street_list = wrap_up_tag_k_v_dict(tag_k_v_dict, 'addr:street')

    for val in list(street_list):
        street_name = val
        street_split = street_name.split()
        if street_split[-1] in directions:
            street_types.add(street_split[-2])
        else:
            street_types.add(street_split[-1])

    return street_types


def wrap_up_tag_k_v_dict(tag_k_v_dict, key):
    '''
    used to look at the key value pairs of nodes, ways, and relations together
    '''
    return tag_k_v_dict['node'][key] | tag_k_v_dict['relation'][key] | tag_k_v_dict['way'][ke


##############################
### Load Data into MongoDB ###
##############################


# Variable maps to swap out non normal values for normal values
# street direction map
ST_DIR_MAP = {'S': 'South',
              's': 'South',
              'South': 'South',
              'E': 'East',
              'e': 'East',
              'East': 'East',
              'W': 'West',
              'w': 'West',
              'West': 'West',
              'N': 'North',
              'n': 'North',
              'North': 'North'}

# Street type map
ST_TYPE_MAP = {'AVenue': 'Avenue',
               'Ave': 'Avenue',
               'Crescent': 'Cresent',
               'Dr': 'Drive',
               'Dr.': 'Drive',
               'Rd': 'Road',
               'St': 'Street',
               'St.': 'Street',
               'Steet': 'Street'}

# Province Map
PROV_MAP = {'ON': 'ON',
            'Ontario': 'ON',
            'on': 'ON',
            'ontario': 'ON'}

# City Map
```

```python
CITY_MAP = {'City of Cambridge': 'Cambridge',
            'City of Kitchener': 'Kitchener',
            'kitchener': 'Kitchener',
            'City of Waterloo': 'Waterloo',
            'waterloo': 'Waterloo',
            'St. Agatha': 'Saint Agatha'}


def map_subin(val, val_map):
    '''
    Subs in a value from a map given the map and a value
    '''
    # returns the maped value if it's in the map
    if val in val_map.keys():
        return val_map[val]
    else:
        # returns the original value if it's not in the map
        return val


def update_street(street):
    '''
    uses the map_subin function to subin corrected street types and street directions
    '''
    # split the street address into a list of words
    st_list = street.split()

    if st_list[-1] in ST_DIR_MAP.keys():
        # if the last word is a direction sub both direction(-1) & type(-2)
        st_list[-1] = map_subin(st_list[-1], ST_DIR_MAP)
        st_list[-2] = map_subin(st_list[-2], ST_TYPE_MAP)
    else:
        # otherwise sub in the street type
        st_list[-1] = map_subin(st_list[-1], ST_TYPE_MAP)

    return ' '.join(st_list)


def update_address(key, val, addr_dict):
    if key == 'addr:street':
        addr_dict[key[5:]] = update_street(val)
    elif key == 'addr:state':
        if not addr_dict.get('province'):
            addr_dict['province'] = map_subin(val, PROV_MAP)
    elif key == 'addr:province':
        addr_dict[key[5:]] = map_subin(val, PROV_MAP)
    elif key == 'addr:city':
        addr_dict[key[5:]] = map_subin(val, CITY_MAP)
    else:
        addr_dict[key[5:]] = val
    return addr_dict


def tag_subtag_process(sub_tag, address, tags):
    key = sub_tag.attrib['k']
    val = sub_tag.attrib['v']

    # addr: tags are sent to update_address function
    if key[0:5] == 'addr:':
        address = update_address(key, val, address)

    # merge 'fixme' and 'FIXME' into 'FIXME'
    elif key in ['fixme', 'FIXME']:
        if tags.get('FIXME'):
            tags['FIXME'] += '\nFIXME: ' + val
        else:
            tags['FIXME'] = val

    # all other tag tags get added as k:v pairs
    else:
        tags[key] = val

    return address, tags


def subtag_process(xml_tree):
    '''
    adds sub tags of an osm xml element to lists and dicts for easy joining to the JSON struc
    '''
    # dicts and lists for constucted values
    node_refs = []
    members = []
    address = {}
    tags = {}

    # looping though each sub tag of xml_tree
    for sub_tag in xml_tree.iter():

        # sub tag of 'tag' sent to tag function
        if sub_tag.tag == 'tag':
            address, tags = tag_subtag_process(sub_tag, address, tags)

        # sub tag of 'nd' appended in order to list
        elif sub_tag.tag == 'nd':
            node_refs.append(int(sub_tag.attrib['ref']))

        # sub tag of 'member' appended in order as a list of dicts
        elif sub_tag.tag == 'member':
            mem = {}
            for key, val in sub_tag.attrib.items():
                if val:
                    if key == 'ref':
                        mem[key] = int(val)
                    else:
                        mem[key] = val
            members.append(mem)

    return node_refs, members, address, tags


def shape_xml_tree(xml_tree):
    '''
    takes an xml element (node, way, or relation) and converts it into a json element includi
    '''
    # returns an empty element if the xml_tree is not a node, way or relation
    if xml_tree.tag not in ['node', 'way', 'relation']:
        return {}

    # This is the element we will return at the end
    element = {}

    # building out the xml_tree attributes
    element['type'] = xml_tree.tag
    element['id'] = int(xml_tree.attrib.get('id'))
    # location info from the start tag is converted into a list of two floats for easy coordi
    if xml_tree.tag == 'node':
        pos = [float(xml_tree.attrib.get('lat')), float(xml_tree.attrib.get('lon'))]
        element['pos'] = pos

    # creation info is saved in a dictionary under the creation key
    element['created'] = {}
    for key, val in xml_tree.attrib.items():
        if key in ["uid", "version", "changeset"]:
            element['created'][key] = int(val)
        if key in ["user", "timestamp"]:
            element['created'][key] = val

    # sub tags are processed in the subtag_process function
    node_refs, members, address, tags = subtag_process(xml_tree)

    # append all the subtag values
    if node_refs:
        element['nd'] = node_refs
    if members:
        element['member'] = members
    if address:
        element['addr'] = address
    if tags:
        element['tag'] = tags

    return element


def process_map(file_in, pretty=False):
    '''
    takes xml file with tag 'node', 'way', or 'relation'

    Unpackes into json compatible dict and list structure.
    saves the json to file for easy import into MongoDB

    {'type':    xml_tree.tag,

     'id':      int(xml_tree('id')),

     'pos':     [float(xml_tree('lat')),
                 float(xml_tree('lon'))],

     'created': {'version':     int(xml_tree('uid')),
                 'changeset':   int(xml_tree('changeset')),
                 'timestamp':   xml_tree('timestamp'),
                 'user':        xml_tree('user'),
                 'uid':         int(xml_tree('uid'))},

     'address': {'housenumber': tag_tag['addr:housenumber'],
                 'postcode': tag_tag['addr:postcode'],
                 'street': tag_tag['addr:street'], ...},

     'member':  [{'type': member_tag('type'),
                  'ref':  int(member_tag('ref')),
                  'role': member_tag('role')},
                 {.........................}],

     'node_refs':[int(nd_tag['ref']),
                  int(nd_tag['ref']), ... ],

     'tag': {tag['k']:  tag_tag['v'],
             tag['k']:  tag_tag['v'],
             ... }
     }
    '''

    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as file_out:
        for _, xml_tree in ET.iterparse(file_in):
            element = shape_xml_tree(xml_tree)
            if element:
                data.append(element)
                if pretty:
                    file_out.write(json.dumps(element, indent=4)+"\n")
                else:
                    file_out.write(json.dumps(element) + "\n")
    return data

if __name__ == '__main__':
    pass

```