P3：整理OpenStreetMap数据

项目综述

本项目选取西班牙城市巴塞罗那作为数据提取的城市。

- OSM文件解压缩之后是244M

巴塞罗那是享誉世界的地中海风光旅游城市和世界著名的历史文化名城，也是我最喜欢的足球队巴塞罗那俱乐部的所在地。这所城市一直是我心目中的足球圣地，我对这座城市的向往和兴趣是我选择这座城市的原因。

数据整理

格式准备

首先导入用到的Python库文件

```python
import xml.etree.cElementTree as ET
import pprint
import re
import json
import codecs
from itertools import chain
from collections import defaultdict, Counter
```

查看该OSM文件中都有哪些tag，每个tag的数量是多少

```python
def count_tags(filename):
```

```
    '''
    Use the iterative parsing to process the map file
and
    find out not only what tags are there, but also
how many.
    '''
    tags = {}
    context = ET.iterparse(filename)
    for event,elem in context:
        if elem.tag not in tags:
            tags[elem.tag] = 1
        else:
            tags[elem.tag] += 1

    return tags
```

Python

定义了三个正则表达式的数据条目，分别为lower、lower_colon和problemchars

1. 如果判定为lower，认为是正常的数据
2. 如果判定为lower_colon， 认为是特殊的数据，需要进行分离处理
3. 如果判定为problemchars， 认为是有问题的数据，需要舍弃或者修正

```
# Check the "k" value for each "<tag>"
# and see if there are any potential problems
lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\.
\t\r\n]')
```

Python

检查分别符合上述正则表达式判定的数据分别有多少

```
def key_type(element, keys):
    if element.tag == "tag":
        if element.get("k") is None:
            pass
        elif lower.match(element.get("k")):
            keys["lower"] += 1
        elif lower_colon.match(element.get("k")):
            keys["lower_colon"] += 1
        elif problemchars.match(element.get("k")):
            keys["problemchars"] += 1
        else:
            keys["other"] += 1
    return keys
```

```
def process_map(filename):
    '''
    process mapfile and return element in file
    '''
    keys = {"lower": 0, "lower_colon": 0,
"problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)
    return keys, element
```

Python

- 结果发现在该数据集中没有匹配problemchars的数据

初步清洗

由于巴塞罗那是一个西班牙的城市，它的街道名称是西班牙语，很多情况下与英语的习惯不相符，因此接下来清理街道的名称这一个项目。

用正则表达式判定街道名称，由于西班牙语中类似"Street"这样的单词通常放在首字母，所以采用下面这样的正则表达式

```
street_type_re = re.compile(r'^\b\S+\.?',
re.IGNORECASE)
```

Python

通过正则表达式判断是否是街道的名称，并且整理街道的类型

```
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)


def is_street_name(element):
    return (element.attrib['k'] == "addr:street")


words_of_street = []
def audit(filename):
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(filename, events=
("start",)):
        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
```

```
                    # Capital the first letter of
every word

audit_street_type(street_types,/
tag.attrib['v'].title())

                    tag =
tag.attrib['v'].title().split()
                    words_of_street.append(tag)
    return street_types
```

Python

首先通过词云展示的方式探索街道名称中单词的使用数目的情况，找出类似英文中"Street"、"Avenue"、"Count"的单词

```
def make_wordcloud(words_of_street):
    '''
    Make wordcloud of tag['v'] by the tag['k'] =
'addr:street'
    '''
    from wordcloud import WordCloud
    from os import path
    d = path.dirname(__file__)

    word_of_street = list(chain(*words_of_street))
    count = Counter(word_of_street)
    street_wordcloud =
WordCloud().generate_from_frequencies (dict(count))
    street_wordcloud.to_file(path.join(d,
"streetmapfreq.jpg"))
```

Python

生成的词云图如下

可以看到在西班牙语中对应的单词有"Carrer"、"Sant"、"Avinguda"、"Balmes"等等

通过词频分析可以找出数据中上述单词的一些错拼或者简拼，一个不完全的mapping

通过词频分析可以找出数据中上述单词的一些错别或者简拼，一个完整的mapping如下所示：

```
mapping = {"Avda":"Avenida",
           "Avda.":"Avenida",
           "Av.":"Avenida",
           "Ave.":"Avenida",
           "Torent":"Torrent",
           "C":"Carrer",
           "C.":"Carrer",
           "C./":"Carrer",
           "C/":"Carrer",
           "Calle":"Carrer",
           "C/Sant":"Carrer",
           "C/Torrassa":"Carrer",
           "Caller":"Carrer",
           "Camí":"Camino",
           "Cami":"Camino",
           "Camp":"Campus",
           "Carretera":"Ctra",
           "Carre":"Carrer",
           "Carrar":"Carrer",
           "Carrier":"Carrer",
           "Cl":"Carrer Del",
           "Cr":"Carrer",
           "Crta":"Ctra",
           "Ctra.":"Ctra",
           "De":"Del",
           "Dels":"Del",
           "Dr.":"Doctor",
           "Passadís":"Passatge",
           "Paseo":"Passeig",
           "Pg":"Passeig",
           "Pg.":"Passeig",
           "Pl":"Placa",
           "Pla":"Placa",
           "Pl,":"Placa",
           "Pº":"Paseo",
           "Ramble":"Rambla",
           "Viaducte":"Viaducto"
           }
```

Python

利用mapping的数据对不干净的街道名称进行清洗

```
def update_name(name, mapping):
    name = name.split()
    for i,j in enumerate(name):
```

```
        if j in mapping:
            name[i] = mapping[j]
    name = " ".join(name)
    return name

def clean_name(element):
    if element.tag == "node" or element.tag == "way":
        for tag in element.iter("tag"):
            if is_street_name(tag):
                tag.attrib['v'] =
update_name(tag.attrib['v'].title())
```

Python

生成文件

将OSM文件中tag为“node”或者“way”的信息提取出来并整理成适和导入MongoDB的格式

```
def shape_element(element):
    '''
    transform the shape of the data into model just
like this:

    {
    "id": "2406124091",
    "type: "node",
    "visible":"true",
    "created": {
              "version":"2",
              "changeset":"17206049",
              "timestamp":"2013-08-03T16:43:42Z",
              "user":"linuxUser16",
              "uid":"1219059"
            },
    "pos": [41.9757030, -87.6921867],
    "address": {
              "housenumber": "5157",
              "postcode": "60625",
              "street": "North Lincoln Ave"
            },
    "amenity": "restaurant",
    "cuisine": "mexican",
    "name": "La Cabana De Don Luis",
    "phone": "1 (773)-271-5176"
    }

    '''
```

```
    node = {}
    if element.tag == "node" or element.tag == "way" :
        node["type"] = element.tag
        created = {}
        addres = {}
        nd = []
        ref = 0
        ad = 0
        pos = 0
        # entry about created
        CREATED = [ "version", "changeset",
"timestamp", "user", "uid"]
        # entry about addres
        ADDRES = ["city",
"housenumber","postcode","street"]
        for attr in element.attrib:
            if attr in CREATED:
                created[attr] = element.get(attr)
            elif attr == "lat":
                pos += 1
                lat = float(element.get(attr))
            elif attr == "lon":
                pos += 1
                lon = float(element.get(attr))
            else :
                node[attr] = element.get(attr)
        for tag in element.iter("nd"):
            ref += 1
            nd.append(tag.get("ref"))
        for tag in element.iter("tag"):
            if problemchars.match(tag.get("k")):
                pass
            elif lower_colon.match(tag.get("k")):
                #addres[tag.get("k").split(":")[1]] =
tag.get("v")
                if tag.get("k").split(":")[1] in
ADDRES:
                    ad += 1
                    if tag.get("k").split(":")[1] ==
"street":
                        addres["street"] =
update_name(tag.get("v"), mapping)
                    else:
                        addres[tag.get("k").split(":")
[1]] = tag.get("v")
```

```
            elif lower.match(tag.get("k")):
                node[tag.get("k")] = tag.get("v")
            else:
                pass
        node["created"] = created
        if ad:
            node["address"] = addres
        if pos:
            node["pos"] = [lat,lon]
        if ref:
            node["node_refs"] = nd
        return node
    else:
        return None
```

Python

生成需要导入MongoDB的json文件

```
def generate_json(file_in, pretty = False):
    '''
    generate json file for importing to MongoDB
    '''
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in
ET.iterparse(file_in.split(".")[0]):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el,
indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return data
```

Python

生成了274M的json文件，将其导入到MongoDB

数据清洗和探索

建立MongoDB数据库并导入数据

```
>mongoimport -d map_of_spain -c barcelona --file
```

```
c:\Users\Amazing\Documents\mapsearch\code\barcelona_spain.osm.json
```

Shell

▶ imported 1281595 documents

数据基础信息

- 节点和路径的数量

```
>db.barcelona.find({"type":"node"}).count()
>db.barcelona.find({"type":"way"}).count()
```

SQL

得到节点的数量为1136303，路径的数量为144889

- 选定某一类型节点（如超市、医院）的数量

得到的结果为

```
{"shop":3852,
 "atm":403,
 "firehydrant":885,
 "public_transport":1465,
 "tram":208,
 "bar":764,
 "restaurant":1817,
 "leisure":8117,
 "cafe":555,,
 "plam":360,
 "parking":807}
```

SQL

数据深入探索

- 探索城市自然、地理环境

1. 巴塞罗那号称"欧洲之花"， 位于地中海沿岸，属于地中海气候，夏季炎热干旱、冬季温和多雨。冬季平均温度为11度，夏季平均温度为24度。
2. 巴塞罗那森林覆盖率高达58.3%， 并且有数量众多、特色鲜明的公园。
3. 巴塞罗那整体属于丘陵地带，城市的许多街区以附近的小丘命名。

接下来将从几个方面对巴塞罗那的数据进行探索

▶ 巴塞罗那的自然环境，用如下语句进行查询

```
> db.barcelona.aggregate([{"$group":

{"_id":"$natural","nums":{"$sum":1}}}])
```

SQL

得到的结果是：

```
{ "_id" : "ridge", "nums" : 2 }
{ "_id" : "valley", "nums" : 43 }
{ "_id" : "gravel", "nums" : 2 }
{ "_id" : "meadow", "nums" : 2 }
{ "_id" : "fell", "nums" : 3 }
{ "_id" : "grass", "nums" : 3 }
{ "_id" : "garden", "nums" : 6 }
{ "_id" : "sand", "nums" : 128 }
{ "_id" : "park", "nums" : 1 }
{ "_id" : "wetland", "nums" : 35 }
{ "_id" : "heath", "nums" : 142 }
{ "_id" : "bare_rock", "nums" : 2 }
{ "_id" : "tree_row", "nums" : 114 }
{ "_id" : "scrub", "nums" : 821 }
{ "_id" : "grassland", "nums" : 658 }
{ "_id" : "water", "nums" : 490 }
{ "_id" : "coastline", "nums" : 73 }
{ "_id" : "Mountain_range", "nums" : 1 }
{ "_id" : "rock", "nums" : 3 }
{ "_id" : "beach", "nums" : 32 }
{ "_id" : "yes", "nums" : 3 }
{ "_id" : "stone", "nums" : 4 }
{ "_id" : "wood", "nums" : 1456 }
{ "_id" : "cave_entrance", "nums" : 162 }
{ "_id" : "cliff", "nums" : 24 }
{ "_id" : "tree", "nums" : 18416 }
{ "_id" : "peak", "nums" : 165 }
{ "_id" : "saddle", "nums" : 13 }
{ "_id" : "spring", "nums" : 86 }
{ "_id" : null, "nums" : 1258705 }
```

SQL

可以看到巴塞罗那的自然景观十分的丰富，不仅种类十分的多样，而且数目也非常多，其中数目最多的是树木（tree），其次是灌木丛（scrub）和草地（grassland）。

可以看到数据中有一个项目是“yes”，我们稍后会对其进行清洗，按照随机分布，树木的占比最多，所以将其清洗为“tree”

- ▶ 探索树木的种类，查询语句如下

```
db.barcelona.aggregate([{"$group":
{"_id":"$leaf_type","nums":{"$sum":1}}}])
```

SQL

结果是：

```
{ "_id" : "broadleaved", "nums" : 174 }
{ "_id" : "needleleaved", "nums" : 137 }
{ "_id" : "mixed", "nums" : 11 }
{ "_id" : null, "nums" : 1281273 }
```

SQL

可见巴塞罗那的数目中既有针叶林，又有阔叶林，两者所占的比重相差不大

- 探索城市人文艺术环境

1. 巴塞罗那是一个历史文化名城，城区内部有很多哥特式建筑、古罗马城墙和中世纪的古老宫殿，其中有八栋建筑被列为世界遗产。
2. 巴塞罗那有为数众多的博物馆和艺术馆，是欧洲艺术宝库中的一颗璀璨明珠。

对巴塞罗那的历史文化景点进行探索，所用查询语句如下：

```
>db.barcelona.aggregate([{"$group":
{"_id":"$historic","nums":{"$sum":1}}}])
```

SQL

结果是：

```
{ "_id" : "church", "nums" : 1 }
{ "_id" : "aqueduct", "nums" : 4 }
{ "_id" : "building", "nums" : 4 }
{ "_id" : "manor", "nums" : 2 }
{ "_id" : "fort", "nums" : 1 }
{ "_id" : "monastery", "nums" : 1 }
{ "_id" : "threshing_floor", "nums" : 1 }
{ "_id" : "industrial", "nums" : 3 }
{ "_id" : "rune_stone", "nums" : 1 }
{ "_id" : "tomb", "nums" : 2 }
{ "_id" : "archaeological_site", "nums" : 29 }
{ "_id" : "castle", "nums" : 18 }
{ "_id" : "elevator", "nums" : 2 }
{ "_id" : "wayside_cross", "nums" : 5 }
{ "_id" : "ruins", "nums" : 114 }
{ "_id" : "citywalls", "nums" : 11 }
{ "_id" : "ship", "nums" : 1 }
{ "_id" : "cannon", "nums" : 9 }
{ "_id" : "yes", "nums" : 23 }
{ "_id" : "farm", "nums" : 1 }
{ "_id" : "palace", "nums" : 1 }
{ "_id" : "memorial", "nums" : 121 }
{ "_id" : "city_gate", "nums" : 1 }
{ "_id" : "monument", "nums" : 119 }
```

```
{ "_id" : null, "nums" : 1281113 }
{ "_id" : "heritage", "nums" : 7 }
```

SQL

可见，数目最多的是纪念碑（monument,memoria，,其次是一些建筑的废墟

在数据中也有"yes"这一项，我们稍后会一并清洗。

- 探索交通情况

巴塞罗那常住人口有161万人，是西班牙的第二大城市，全市面积总共101.9平方公里，是世界上人口最稠密的城市之一，因此城市交通必定是一个十分重要的问题。

首先对它的高速公路情况进行探索：

```
> db.barcelona.aggregate([{"$group":
{"_id":"$highway","nums":{"$sum":1}}}])
```

SQL

得到的结果是：

```
{ "_id" : "corridor", "nums" : 5 }
{ "_id" : "proposed", "nums" : 19 }
{ "_id" : "construction", "nums" : 109 }
{ "_id" : "motorway_link", "nums" : 666 }
{ "_id" : "motorway", "nums" : 781 }
{ "_id" : "primary_link", "nums" : 487 }
{ "_id" : "road", "nums" : 14 }
{ "_id" : "unclassified", "nums" : 1184 }
{ "_id" : "path", "nums" : 4152 }
{ "_id" : "track", "nums" : 4289 }
{ "_id" : "tertiary_link", "nums" : 687 }
{ "_id" : "bridleway", "nums" : 46 }
{ "_id" : "secondary_link", "nums" : 479 }
{ "_id" : "pedestrian", "nums" : 3245 }
{ "_id" : "secondary", "nums" : 1616 }
{ "_id" : "primary", "nums" : 674 }
{ "_id" : "residential", "nums" : 20422 }
{ "_id" : "tertiary", "nums" : 3759 }
{ "_id" : "service", "nums" : 7179 }
{ "_id" : "traffic_mirror", "nums" : 1 }
{ "_id" : "platform", "nums" : 55 }
{ "_id" : "steps", "nums" : 2270 }
{ "_id" : "living_street", "nums" : 1367 }
{ "_id" : "raceway", "nums" : 16 }
{ "_id" : "cycleway", "nums" : 1053 }
{ "_id" : "elevator", "nums" : 112 }
{ "_id" : "rest_area", "nums" : 14 }
```

```
{ "_id" : "turning_loop", "nums" : 2 }
{ "_id" : "services", "nums" : 15 }
{ "_id" : "yes", "nums" : 3 }
{ "_id" : "milestone", "nums" : 7 }
{ "_id" : "mini_roundabout", "nums" : 50 }
{ "_id" : "bus_stop", "nums" : 2055 }
{ "_id" : "speed_camera", "nums" : 136 }
{ "_id" : "passing_place", "nums" : 4 }
{ "_id" : "incline", "nums" : 1 }
{ "_id" : "trunk", "nums" : 659 }
{ "_id" : "give_way", "nums" : 319 }
{ "_id" : "stop", "nums" : 202 }
{ "_id" : "footway", "nums" : 7240 }
```

SQL

得到的结果不是很令人理解，可以看到条目显示了类似住宅区、人行道、服务区、车站等的信息数目。隐约可以感受到巴塞罗那的交通应该比较发达。

对道路的最大限速情况进行探索：

```
>db.autos.find({"maxspeed":
{"$gt":"70","$lte":"80"}}).count()
```

SQL

得到的结果是：

```
-50:5850|50-60:369|60-70:30|70-80:856
|80-90:94|90-100:0|100-120:349|120-150:6
```

SQL

速度主要集中在50-60这样的街区低速区和100-120这样的公路高速区

数据再次清洗

- 对natural中的“yes”条目清洗为“tree”

```
> db.barcelona.update({"natural":"yes"},{$set:
{"natural":"tree"}})
```

SQL

- 将historic中的“yes”条目删去

```
> db.barcelona.remove({"historic":"yes"})
```

SQL

- 对电话号码（phone）的格式进行清洗

通过查询操作发现数据信息里面的电话号码这一个项目格式有些混乱，有如下格式

```
'+34 935729026'
'+34935677447'
'934417526'
'+34 934 73 41 28'
'93 466 38 38'
'678 60 45 23'
'934 511 786'
'+34 933 429 628'
'+34 93 232 67 60'
'+34 932036349'
'(+34) 93 310 79 61'
```

SQL

+34 是西班牙的国家电话区号，可以全部省去，可以看到剩下的电话是9维数字，为方便起见，我们将其清洗成

'934417526'这样的形式。

```
def phone_clean(db):
  """
  clean phone_numbers to format just like '934417526'
  """
  documents = db.barcelona.find({"phone":
{"$exists":1}})
  for document in documents:
    phone = document["phone"]
    if "(" in phone:
      phone = phone.replace("(", "")
      phone = phone.replace(")", "")
    if "+34" in phone:
      phone = phone.replace("+34", "")
    if phone[:2] == "34":
      phone = phone.replace("34", "")
    if "-" in phone:
      phone = phone.replace("-", "")
    if "+" in phone:
      phone = phone.replace("+", "")
    phone = "".join(phone.split())
    if "." in phone:
      phone = "".join(phone.split("."))
    db.barcelona.update({"phone":document["phone"]},
{"$set":{"phone":phone}})
```

Python

清理完成后电话号码这一项数据的格式已经基本统一。

数据总览及额外建议

这次的数据探索尝试着从几个方面得出了一些结论，但仍然有很多的信息可以挖掘。

- 遇到的问题

由于巴塞罗那是西班牙城市，有很多信息都是用西班牙语进行描述的，这部分信息很难提取出来

很多信息像电话号码、邮编等信息格式非常不统一，需要对其进行清洗才能提取有效的信息

- 取得的成果

初步了解了巴塞罗那自然地理和人文艺术的相关信息，并对城市的基本信息进行了初步的探索

- 额外的建议

☐ 由于数据中邮编的格式前后也有不统一的地方，因此可以用正则表达式对邮编进行清洗

▶ 这样可以得到更加干净的数据信息，方便以后的应用

☐ 对经纬度进行分块处理，统计各个地区的如餐厅、艺术馆等的分布情况

▶ 对经纬度分块处理可以获得更多的信息，比如可以通过判断餐厅、超市的密度判断城市的商业区或居民区。但是这一项操作相对比较困难，很容易将分块区域太大或者太小。并且只有node这个标签里才有经纬度信息，也可能会造成信息提取的不全面。