Data wrangling with mongoDB

Data Wrangling process involves the following steps:

- Gathering
- Extracting
- cleaning
- storing the data

Analysis of the data is carried out after these steps.

Gathering the data

- In this project the data was gathered from the website openstreetmap.org,as per the instructions.
- "The uncompressed size of the file must be atleast 50MB " was the criteria given to chose a data.
- The compressed data file was downloaded and found that they were very large in an uncompressed format .
- The data of fairly small size was selected to avoid the long execution times for each query.

Extracting the data

- The data was extracted from the metro extracts section of the openstreetmap.org. The data was in compressed format.
- The compressed data was then unzipped to get the actual data of size greater than 50MB.
- The part of the world ,chosen for this project is Southampton of England. The uncompressed size of the file is about 63MB.

File size

Southampton.osm 63.0 MB

Southampton.osm.bz2 4.5 MB

Cleaning the data

- Data cleaning is an iterative process involves identifying the problems and fix them.
- The data may contain missing or extra fields ,that needs to be fixed.
- Json document may not be in the expected structure,that needs to be fixed.

Data cleaning -Problems encountered in the map

- Before adding the data into the MongoDB, data has to be processed and cleaned.The various tags were identified and counted.
- The data needs to be explored through tags for any invalid or problematic tags. The problematic tags were identified and corrected in the next steps.

1)over-abbreviated street names

- The data had over-abbreviated street names. For example : st,st. and St were used to refer the streets . S. and N. are abbrevations for South and North respectively.
- The street types were gathered to identify the over-abbreviated street names.
- The full form of the abbreviation was used to overcome this.

A part of over-abbreviated street names coding:

def test():

```
    st_types = audit(filename)

    pprint.pprint(dict(st_types))


    for st_type, ways in st_types.iteritems():
        for name in ways:
            better_name = update_name(name, mapping)
            print name, "=>", better_name
```

test()

A part of the output:

Old Cricket Mews => Old Cricket Mews Grange Rd => Grange Road Hythe Rd => Hythe Road Redbridge Hill => Redbridge Hill Abbey Hill => Abbey Hill

2)Postal codes

- The data had issues in the postal codes field.
- The postal codes of southampton were described as - the outward postcode page 'EC2M' will give you an alphabetical list covering all postcodes located in the region, from 'EC2M 1BB' through to 'EC2M 7YA'. Most of the postal codes were in the expected format(eg :'SO16 1BB')
- some postal codes were in lowercase ,lacking of a space right after outward postcode.
- ["So40 4wr","so16 8hy","SO172FZ"] were samples of the erroreous postal codes.
- The erroreous postal codes were fixed and specified pattern of postal codes was followed there after.

A part of postal codes update coding:

postalcode = ["SO14","SO15","SO16","SO17","SO18","SO19","SO20","SO21","SO22","SO23","SO24","SO30","SO31 "SO41","SO42","SO43","SO44","SO45","SO50","SO51","SO52","SO53"]

def update_postcode(postc):

```
    value= postc.lower()
    match = re.search(r'so\d\d\s\d\w\w', value)
    if match:
        return value.upper()
    elif len(value)<8:
        tune= value[:4] +' ' + value[4:]
        return tune.upper()
    else:
```

```
    else:
        return "error"
```

print update_postcode("So40 4wr") print update_postcode("so16 8hy") print update_postcode("SO172FZ") print update_postcode("SO16 8HY")

A part of the output:

SO40 4WR SO16 8HY SO17 2FZ SO16 8HY

3)Data Model

- Before adding the cleaned data into the MongoDB, the data modeling needs to be done.
- The given data was converted to the preferred data model ,to insert into the MongoDB.
- Once the preferred data model was obtained and saved as a json file.
- A json data can have different fields,nested arrays or even nested objects.
- The Json data will be then added to the MongoDB.

File size

Southampton.osm 63.0 MB

Southampton.osm.json 63.6 MB

when data cleaning is carried out ,it is clear that the dataset of southampton is incomplete. print data[0]

{'id': '132707', 'visible': None, 'type': 'node', 'pos': [50.9454657, -1.4775675], 'created': {'uid': '260682', 'changeset': '8139974', 'version': '5', 'user': 'monxton', 'timestamp': '2011-05-14T11:45:29Z'}}

print data[-5]

{'node_refs': ['4464031263', '4464031264', '4464031265', '4464031266', '4464031263']}

print data[-2000]

{'node_refs': ['3801637140', '3801637142', '3801637143', '3801637141', '3801637140'], 'address': {'street': 'Oswald Road', 'postcode': 'SO19 9SQ'}}

print data[-1] {'amenity': 'restaurant', 'node_refs': ['4479585324', '4479585325', '4479585326', '4479585327', '4479585324'], 'name': 'Cove', 'address': {'street': 'Shamrock Quay'}}

In the above output,it is clear that

- some data lacks the address field
- some data lacks postal code in the address field
- some data lacks there are no name field

when a dataset misses certain fields ,the people's interest to use this map will start decreasing.This southampton map is incomparable with Google maps or bing maps ,which has pretty much large details.

Storing the data

- Creating a local host connection with MongoDB and a database named "maps" was created

- Creating a local host connection with MongoDB and a database named "maps" was created .
- Pymongo (a library) was used to connect to the database.
- The data was uploaded into the db.S_E_city where S_E_city is the collection.
- various queries can be created and run on the database to get required information.

A part of the code:

client = MongoClient('localhost:27017') db = client.maps

print db.S_E_city

output:

Collection(Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), u'maps'), u'S_E_city')

Analysis of the data

-Simple queries were run to get the counts.

db.S_E_city.find().count()

output: 319928

query: db.S_E_city.find({"created.user":"monxton"}).count()

output:

1750

- A query was generated to get the results for top 20 users who created most posts.

print "The top 20 users who created most posts are " query = [{"group":{"_id":"created.user", "count":{"sum":1}}},{"sort":{"count":-1}},{"$limit":20}] results = db.S_E_city.aggregate(query) printtheresult(results)

- A query was run on DB to get the top 10 postcodes based on the counts.

print "The top 10 postcodes based on the counts are" query = [{"match":{"address.postcode":{"exists":1}}}, {"group":{"_id":"address.postcode", "count":{"sum":1}}},{"sort":{"count":-1}},{"$limit" : 10}] results = db.S_E_city.aggregate(query) printtheresult(results)

- A query was run on DB to get the data having the postcode of "SO17 1QG".

query = [{"$match":{"address.postcode":"SO17 1QG"}},

```
{"$sort":{"count":-1}}]
```

results = db.S_E_city.aggregate(query) printtheresult(results)

- A query was generated to get the most used outward postcode query = [{"match":{"address.postcode":{"exists":1}}},

```
{"$project":{"address.street":1,
"outward postcode": { "$substr": [ "$address.postcode", 0,
4 ] }}},
{"$group":{"_id":{"postcode":"$outward postcode"},
```

```
"count":{"$sum" :1}}},

{"$sort":{"count":-1}},
{"$limit":10}]
```

- A query was generated to get the unique street names in the given outward postcode.

```
project":{"address.street":1,  address.postcode", 0, 4 ] }}},
query = [{"  "outward postcode": { "substr": [ "  {"  match":{"outward
postcode":{"$eq":"SO17"}}},

{"$group":{"_id":{"postcode":"$outward postcode"},
"unique_street":{"$addToSet" :"$address.street"},
"street_count":{"$sum" :1}}}

]
```

- A query was generated to get the number of unique streets in the given outwardpostcode.

```
project":{"address.street":1,  address.postcode", 0, 4 ] }}},
query = [{"  "outward postcode": { "substr": [ "  {"  match":{"outward
postcode":{"$eq":"SO17"}}},

{"$group":{"_id":{"postcode":"$outward postcode"},
"unique_street":{"$addToSet" :"$address.street"}
}},
{ "$unwind": "$unique_street" },
{"$group": {"_id":{"postcode":"SO17"},"unique_street_count": {
"$sum": 1 }}}
]
```

- A query was generated to find the top 5 contributors with their contribution ratio.The contribution ratio is calculated by ratio of individual contribution and total contribution.

```
total_count = db.S_E_city.find({'type' :'node'}).count() query = [{"group":{"_id":"created.user", "count":
 sum":1}}},
{"  {"  project":{"Contribution ratio": {"multiply":[{"divide":["$count",total_count]},100] }}},

{"$sort":{"Contribution ratio": -1}},{"$limit":5}]

results = db.S_E_city.aggregate(query) printtheresult(results)
```

output: {u'_id': u'Chris Baines', u'Contribution ratio': 35.96109984324039} {u'_id': None, u'Contribution ratio': 18.843099234032437} {u'_id': u'0123456789', u'Contribution ratio': 8.145927593405695} {u'_id': u'Harjit (CabMyRide)', u'Contribution ratio': 6.545270837512351} {u'_id': u'Nick Austin', u'Contribution ratio': 6.228779875335251}

The name of second top contributor is not specified in the dataset.

Additional ideas

The Southampton of England is fairly small when compared to other more active cities in

OpenStreetMap. The original data was relatively clean and only few problems were encountered in the map. It is fairly easy to create a robust cleaning algorithm with MongoDB. The robust cleaning algorithms, and running the scripts regularly will clean the data automatically.

when a dataset misses certain fields and values ,this will impact on people's interest to use this map .The southampton osm map is incomparable with Google maps or bing maps (which contains pretty much large details). The map could be improved by adding both missing fields and extra fields based on people's need.

periodic update of the map ,helps in having the latest information on the map .when the periodic updates happen, the size of the dataset increases which may hinder the loadingtime of the data but it'll definetely help the mapusers. A very powerful processor ,server can make the processing more faster.