Problems encountered in data:

As an initial step, I wrote functions in python (in audit_islamabad.py) that printed unique values of street addresses, phone numbers and postcodes from the osm file. The following problems were identified after reviewing these functions' outputs and later cleaned using functions in the 'shape_data.py' file.

Inconsistencies in street addresses:

1. Street numbers were mentioned in a variety of different ways (including typographical errors such as `'streety'`). I made all the street names consistent so that all addresses follow the following pattern:
   (**Street *Number*** e.g '**Street 11', 'Street 55A'** etc)
2. Roads were mentioned using a variety of strings (eg. `'rd', 'Rd', 'ROAD'`). For uniformity in street addresses, all such variations were substituted by `'Road'.`
3. Most of Islamabad's area is divided into sectors. Upon auditing the data, I found out that part of street addresses contained sector names but the rest didn't (even though their corresponding full addresses contained sector names). Therefore to maintain uniformity, I fetched the street addresses from the full address fields and added them to the corresponding street addresses that did not contain sector names.
4. Format of the sector names were inconsistent at places. Most of the sector names in the data followed the following pattern `r'[D-I]-[0-9]/[1-4]?'` **(e.g F-6/3)** where the numbers ranging from 1 to 4 represent a sub-sector and are optional. Some of the inconsistencies in sector name formats (such as I8 or I/8/3) were addressed using regular expressions.

Inconsistencies in phone numbers and postal codes:

1. Phone numbers in the data followed multiple patterns. This task was challenging as there was a lot of variation in area codes and mobile phone codes (The metro extract data also contained data from twin city Rawalpindi and a few other nearby cities). Moreover some fields had multiple phone numbers. A function `std_phone_no()` was written to address this problem and standardize all phone numbers so that they follow the pattern (country code-area code-phone no. e.g +92-51-2852024). Multiple phone numbers were saved in an array such as `[u'+92-51-8313200', u'+92-51-5766380', u'+92-51-5590063']`
2. A variety of postal codes were found. Postal codes of surrounding cities mentioned in the following table were also found. After shaping the data and putting it on MongoDB, querying revealed the following count of postal codes in the data:

Post code	City	Count
44000	Islamabad	21
46000	Rawalpindi	30
46060	Rawalpindi	1

46600	Rawalpindi	2
22010	Abbotabad	2
10530	Mirpur AJK	3
4600	Typo (Rawalpindi)	1

4600 is an invalid postcode (probably a typographical error) which was updated by 46000 by querying the document with erroneous postcode and replacing it using **`save()`** method.

Overview of the data:

Some of the basic facts and figures about the data are given in this section

Data size:

islamabad_pakistan.osm 63199 kb
islamabad_pakistan.osm.json 97453 kb [pretty formatting]

Unique user count:

The following result was obtained after using the count operator in the aggregation method over the set of users:
{u'_id': u'user', u'count': 356}

Rest of the features:

```
In [26]: unique_users = db.islamabad.aggregate([{"$group" : {"_id" : "user", "user" : {"$addToSet" : "$created.user"}}},
                                                {"$unwind" : "$user"},
                                                {"$group" : {"_id" : "user", "count" : {"$sum" : 1}}}])
         print_aggregation(unique_users,0)

         {u'_id': u'user', u'count': 356}

In [27]: db.islamabad.find({"type" : "node"}).count()
Out[27]: 314514

In [28]: db.islamabad.find({"type" : "way"}).count()
Out[28]: 29540

In [29]: db.islamabad.find({"amenity" : "cafe"}).count()
Out[29]: 27

In [30]: db.islamabad.find({"amenity" : "school"}).count()
Out[30]: 311
```

From the ipython notebook snapshot above, a few data features are listed below:

1. **Unique Users:** 356
2. **Number of nodes:** 314514
3. **Number of ways:** 29540
4. **Number of cafes:** 27
5. **Number of schools:** 311

Additional ideas about the data:

An open source tourism application can be developed on top of the osm data that enables the tourists and local residents to search for cafes, restaurants (of a particular cuisine), hospitals, schools and other major types of places from the 'amenities' field. This application can serve as a motivation for the business community to contribute to the osm data about their businesses so that they can improve their customer base and popularity.

The application can have a feature where the user/ tourist requests for directions to a particular destination business outlet from a particular landmark. The business owners can respond these requests by specifying a route using existing OSM data. If the data that describes the route is absent, the business owners can add the landmark and street information to the OSM data (by using OSM editor).The benefit of this activity will be that a lot of new street and landmark information will be added to the existing OSM data of the city. GPS and navigation capability can also be added to help the tourists reach their destinations.

The possible challenges associated to such an application are:

1. response time to a request
2. maintaining the quality of data contributed.

For the second challenge, a rating system can be devised to flag routes that were not helpful to the users / tourists. The data associated to these routes can be reviewed to check for errors.

Conclusion:

During this project I was able to successfully audit the various aspects of the data and clean street addresses, phone numbers and postcodes to maintain uniformity and consistency. The data was later put on mongoDB and queries were run to obtain its various features. I also discovered that the metro extract of islamabad also contained data from surrounding cities. For the purposes of this project, i did not get rid of this data but that can also be achieved using the remove() method available in mongoDB.