Using MongoDB to Retrieve Information ¶

In this notebook, we will look at the `PyMongo` library, and perform some common tasks in MongoDB with it. We will be making use of data from food.gov.uk (http://ratings.food.gov.uk/), which gives information about the hygiene ratings of all food establishments in the country.

The code required to connect to the database is as follows:

(*for more information (http://mongodb.github.io/casbah/3.1/reference/connecting/) about connecting to MongDB*)

In [6]:

```
# You don't need to write anything here
from pymongo import MongoClient

client = MongoClient('mongodb://cpduser:M13pV5woDW@mongodb/health_data')
db = client.health_data
```

The MongoDB querying language is very similar to JavaScript, and in Python we make use of **dictionaries** to get the appropriate name/value pairs.

WARNING! Make sure you are careful when you run your queries. If you try and get all 500,000 records displaying on the page it will take a while and could well crash your browser!

Just like using the native Mongo client, you can run functions or access a collection from the query object by using dot notation, so the `uk` collection would be `db.uk`. You can also use `db['uk']`, which can be more useful, e.g., if you are using variable names to access the different collections.

There is a **function** called `collection_names` (http://api.mongodb.com/python/2.5/tutorial.html) which can be performed on the database. Run this function to see the names of the collections in the database.

In [5]:

```
# YOUR CODE HERE
db.collection_names()
```

```
['derbyshire_dales', 'north_kesteven', 'causeway_coast_and_glens', 'sl
ough', 'east_renfrewshire', 'taunton_deane', 'warrington', 'south_lana
rkshire', 'torfaen', 'south_ribble', 'wycombe', 'forest_heath', 'erewa
sh', 'nuneaton_and_bedworth', 'rotherham', 'south_norfolk', 'wirral',
'torbay', 'newcastle_upon_tyne', 'oadby_and_wigston', 'west_lothian',
'epsom_and_ewell', 'pendle', 'bridgend', 'derry_city_and_strabane', 'r
ibble_valley', 'lichfield', 'south_hams', 'mid_sussex', 'test_valley',
'hertsmere', 'bristol', 'woking', 'aberdeen_city', 'armagh_city,_banbr
idge_and_craigavon', "king's_lynn_and_west_norfolk", 'rugby', 'purbec
k', 'powys', 'newark_and_sherwood', 'plymouth_city', 'chelmsford', 'ha
mmersmith_and_fulham', 'west_oxfordshire', 'southampton', 'hackney',
'east_hertfordshire', 'tendring', 'enfield', 'rushmoor', 'eastbourne',
'eden', 'bolton', 'tower_hamlets', 'west_lindsey', 'dover', 'south_som
erset', 'antrim_and_newtownabbey', 'ashford', 'dartford', 'east_hampsh
ire', 'mendip', 'harrow', 'maidstone', 'conwy', 'swale', 'harborough',
'durham', 'worcester_city', 'brighton_and_hove', 'milton_keynes', 'bir
mingham', 'sheffield', 'luton', 'rother', 'aberdeenshire', 'pembrokesh
ire', 'southend-on-sea', 'north_east_lincolnshire', 'fermanagh_and_oma
gh', 'north_devon', 'isle_of_wight', 'north_hertfordshire', 'shepway',
'tandridge', 'lewisham', 'east_devon', 'blackpool', 'north_west_leices
tershire', 'city_of_london_corporation', 'edinburgh_(city_of)', 'west_
dorset', 'lincoln_city', 'rochford', 'liverpool', 'sandwell', 'argyll_
and_bute', 'wolverhampton', 'wokingham', 'chorley', 'brent', 'cotswol
d', 'leicester_city', 'blaby', 'croydon', 'south_oxfordshire', 'darlin
gton', 'telford_and_wrekin_council', 'north_tyneside', 'dacorum', 'bol
sover', 'braintree', 'glasgow_city', 'isles_of_scilly', 'suffolk_coast
al', 'havering', 'angus', 'copeland', 'thanet', 'richmondshire', 'oldh
am', 'gravesham', 'midlothian', 'melton', 'worthing', 'peterborough_ci
ty', 'newry,_mourne_and_down', 'north_dorset', 'wealden', 'gwynedd',
'high_peak', 'south_kesteven', 'kirklees', 'north_east_derbyshire', 'h
arrogate', 'exeter_city', 'sefton', 'broxbourne', 'carmarthenshire',
'south_lakeland', 'york', 'three_rivers', 'cardiff', 'crawley', 'inver
clyde', 'kettering', 'redcar_and_cleveland', 'derby_city', 'east_ridin
g_of_yorkshire', 'lambeth', 'stockport', 'burnley', 'newcastle-under-l
yme', 'colchester', 'new_forest', 'clackmannanshire', 'fenland', 'cher
well', 'hartlepool', 'newport', 'winchester_city', 'nottingham_city',
'flintshire', 'weymouth_and_portland', 'perth_and_kinross', 'bournemou
th', 'blaenau_gwent', 'watford', 'north_warwickshire', 'northumberlan
d', 'north_lanarkshire', 'basingstoke_and_deane', 'fylde', 'gloucester
_city', 'portsmouth', 'vale_of_white_horse', 'wandsworth', 'wychavon',
'reading', 'tameside', 'east_dunbartonshire', 'huntingdonshire', 'isli
ngton', 'wigan', 'east_northamptonshire', 'gosport', 'middlesbrough',
'lancaster_city', 'rossendale', 'amber_valley', 'wrexham', 'preston',
'medway', 'poole', 'manchester', 'rushcliffe', 'mansfield', 'tonbridge
_and_malling', 'cheshire_east', 'tewkesbury', 'sevenoaks', 'wiltshir
e', 'carlisle_city', 'shetland_islands', 'swindon', 'central_bedfordsh
ire', 'rochdale', 'chiltern', 'falkirk', 'neath_port_talbot', 'east_ca
mbridgeshire', 'scottish_borders', 'walsall', 'boston', 'wyre_forest',
'redbridge', 'chichester', 'tamworth', 'stratford-on-avon', 'shropshir
e', 'babergh', 'bassetlaw', 'east_staffordshire', 'castle_point', 'kno
wsley', 'bedford', 'oxford_city', 'bury', 'norwich_city', 'swansea',
'halton', 'north_norfolk', 'south_buckinghamshire', 'mid_and_east_antr
im', 'torridge', 'sedgemoor', 'tunbridge_wells', 'stafford', 'windsor_
and_maidenhead', 'east_dorset', 'stockton_on_tees', 'fareham', 'north_
lincolnshire', 'denbighshire', 'malvern_hills', 'rhondda_cynon_taf',
'uttlesford', 'st_albans_city', 'ceredigion', 'havant', 'dundee_city',
```

```
'newham', 'cambridge_city', 'north_ayrshire', 'stevenage', 'west_somer
set', 'wyre', 'mid_suffolk', 'waltham_forest', 'hart', 'st_helens', 'b
elfast_city', 'kensington_and_chelsea', 'barnsley', 'northampton', 'wa
veney', 'barking_and_dagenham', 'south_holland', 'christchurch', 'barn
et', 'brentwood', 'highland', 'sunderland', 'anglesey', 'herefordshir
e', 'wakefield', 'stirling', 'bradford', 'merthyr_tydfil', 'hillingdo
n', 'ashfield', 'arun', 'craven', 'solihull', 'broxtowe', 'sutton', 'c
annock_chase', 'aylesbury_vale', 'south_tyneside', 'hull_city', 'gates
head', 'guildford', 'hambleton', 'south_derbyshire', 'epping_forest',
'redditch', 'st_edmundsbury', 'adur', 'fife', 'south_northamptonshir
e', 'vale_of_glamorgan', 'haringey', 'runnymede', 'bexley', 'cheshire_
west_and_chester', 'east_ayrshire', 'lisburn_and_castlereagh_city', 'm
id_ulster', 'spelthorne', 'dumfries_and_galloway', 'kingston-upon-tham
es', 'selby', 'west_lancashire', 'south_cambridgeshire', 'greenwich',
'orkney_islands', 'cheltenham', 'ipswich', 'barrow-in-furness', 'river
_tees', 'salford', 'scarborough', 'bromley', 'ryedale', 'south_staffor
dshire', 'forest_of_dean', 'stoke-on-trent', 'trafford', 'hounslow',
'bath_and_north_east_somerset', 'south_gloucestershire', 'cornwall',
'westminster', 'broadland', 'corby', 'surrey_heath', 'eastleigh', 'wes
t_devon', 'comhairle_nan_eilean_siar_(western_isles)', 'gedling', 'cae
rphilly', 'harlow', 'renfrewshire', 'bracknell_forest', 'hinckley_and_
bosworth', 'thurrock', 'chesterfield', 'east_lindsey', 'daventry', 'le
wes', 'camden', 'blackburn', 'charnwood', 'maldon', 'hyndburn', 'west_
dunbartonshire', 'mid_devon', 'stroud', 'merton', 'monmouthshire', 'no
rth_somerset', 'uk', 'calderdale', 'southwark', 'hastings', 'reigate_a
nd_banstead', 'allerdale', 'doncaster', 'basildon', 'horsham', 'richmo
nd-upon-thames', 'wellingborough', 'breckland', 'leeds', 'warwick', 'b
romsgrove', 'mole_valley', 'canterbury_city', 'dudley', 'elmbridge',
'great_yarmouth', 'staffordshire_moorlands', 'moray', 'ealing', 'ards_
and_north_down', 'east_lothian', 'coventry', 'waverley', 'welwyn_hatfi
eld', 'west_berkshire', 'teignbridge', 'south_ayrshire']
```

Querying

Querying is done on collection objects. Start with using the `find_one` function on any collection to investigate the structure of the data.

In [6]:

```
# You don't need to write anything here
db['uk'].find_one()
```

Out[6]:


```
{'AddressLine2': '16a Adelphi Street',
 'AddressLine3': 'Preston',
 'BusinessName': '3 Monkeys Sandwich Bar',
 'BusinessType': 'Restaurant/Cafe/Canteen',
 'BusinessTypeID': '1',
 'ConfidenceInManagement': 10,
 'FHRSID': '90105',
 'Geocode': {'coordinates': [-2.706293, 53.763151], 'type': 'Point'},
 'Hygiene': 10,
 'Lat': 53.763151,
 'Lng': -2.706293,
 'LocalAuthorityBusinessID': '244',
 'LocalAuthorityCode': '202',
 'LocalAuthorityEmailAddress': 'info@preston.gov.uk',
 'LocalAuthorityName': 'Preston',
 'LocalAuthorityWebSite': 'http://www.preston.gov.uk',
 'NewRatingPending': 'False',
 'PostCode': 'PR1 7BE',
```


It can be useful to run the `find_one` function when you are trying a certain set of search conditions, to check that you are getting the results you expect. To add conditions to a query, the first parameter of the function is a dictionary in the format `{'field': 'value'}`. Search for the first document which has a `Region` value of `'london'`

In [ ]:

```
# YOUR CODE HERE
print(db.uk.find_one({'Region': 'london'}))
```

Query Operators

In addition to searching for equality, there are a range of operators (https://docs.mongodb.com/manual/reference/operator/query/) which can be used in MongoDB, such as `$lt` for less than, `$gte` for greater than or equal to, etc.

In this case, PyMongo is slightly different to the native Mongo client. For PyMongo, the query is written `{field_name: {'$eq': 5} }`. Remember that these operators need to be strings.

You'll notice that there is a dictionary inside a dictionary in that query. This is normal, and something we'll see a lot of!

Write a query to find the first business in Southampton which has a `RatingValue` of less than 5.

In [3]:

```
# YOUR CODE HERE
print(db.southampton.find_one({'RatingValue': {'$lt': 5}}))
```

```
{'BusinessType': 'Restaurant/Cafe/Canteen', 'Geocode': {'type': 'Poin
t', 'coordinates': [-1.395055, 50.922154]}, 'LocalAuthorityName': 'Sou
thampton', 'LocalAuthorityCode': '877', 'Hygiene': 10, 'FHRSID': '7060
71', 'LocalAuthorityWebSite': 'http://www.southampton.gov.uk', 'Lng':
-1.395055, 'RatingKey': 'fhrs_3_en-GB', 'SchemeType': 'FHRS', 'Lat': 5
0.922154, 'RatingDate': datetime.datetime(2016, 2, 16, 0, 0), 'Busines
sTypeID': '1', '_id': ObjectId('5be5463dc4cc3a0001cb266b'), 'Region':
'south_east', 'PostCode': 'SO17 2FW', 'ConfidenceInManagement': 10, 'N
ewRatingPending': 'False', 'AddressLine1': '110 Portswood Road', 'Rati
ngValue': 3, 'Structural': 10, 'AddressLine2': 'Southampton', 'LocalAu
thorityEmailAddress': 'hygiene.rating@southampton.gov.uk', 'LocalAutho
rityBusinessID': '14930/0110/0/000', 'BusinessName': '7 Bone Burger C
o', 'Scores': {'Hygiene': 10, 'Structural': 10, 'ConfidenceInManagemen
t': 10}}
```

Returning Part of a Document

By default, all values in a document will be returned from a query. This is not always the desired outcome, so it is possible to modify which parts of the document are returned. This is done by the optional second parameter to a `find` or `find_one` query as a dictionary in the format `{"keep_this_field": 1, "ignore_this_field": 0}`.

If this parameter exists, then any field name which is not specified will not be returned unless specifically requested. For example, consider the code below, which returns the name of the first business from Aberdeenshire:

In [13]:

```
db.aberdeenshire.find_one({}, {'BusinessName': 1})
```

Out[13]:

```
{'BusinessName': '2nd Dimensions', '_id': ObjectId('5be54625c4cc3a0001
c9f943')}
```

In [11]:

```
db.aberdeenshire.find_one()
```

Out[11]:

```
{'AddressLine2': 'Turriff',
 'AddressLine3': 'Aberdeenshire',
 'AddressLine4': 'AB53 4DX',
 'BusinessName': '2nd Dimensions',
 'BusinessType': 'Retailers - other',
 'BusinessTypeID': '4613',
 'FHRSID': '75064',
 'Geocode': None,
 'LocalAuthorityBusinessID': '17437',
 'LocalAuthorityCode': '761',
 'LocalAuthorityEmailAddress': 'environmental@aberdeenshire.gov.uk',
 'LocalAuthorityName': 'Aberdeenshire',
 'LocalAuthorityWebSite': 'http://www.aberdeenshire.gov.uk/',
 'NewRatingPending': 'False',
 'RatingDate': datetime.datetime(2013, 2, 14, 0, 0),
 'RatingKey': 'fhis_pass_en-GB',
 'RatingValue': None,
 'Region': 'scotland',
 'SchemeType': 'FHIS',
 'Scores': None,
 '_id': ObjectId('5be54625c4cc3a0001c9f943')}
```

There are three things to notice about this query.

1. Firstly, the dictionary as the first parameter is empty, meaning that there are no criteria for the search result.
2. The `BusinessName` field is returned as expected
3. The `_id` field is also returned without our asking for it! This is an exception to the rule of requiring to request a field specifically. In order to avoid having this field (and you will need to do this for the visualisation exercise, because having it causes problems for the `Bokeh` library), you simply request that it is not there, as in the code below:

In [14]:

```
db.aberdeenshire.find_one({}, {'BusinessName': 1, '_id': 0})
```

Out[14]:

```
{'BusinessName': '2nd Dimensions'}
```

Test Yourself

Write a query to return the `BusinessType` of the first business in Swansea with a `RatingValue` of 1, excluding the `_id`

In [20]:

```
# YOUR CODE HERE

db.swansea.find_one({'RatingValue': 1}, {'BusinessType': 1, '_id': 0})
```

Out[20]:

```
{'BusinessType': 'Retailers - other'}
```

Cursors

Whereas `find_one` returns a single record, the `find` method returns a `Cursor` (http://api.mongodb.com/python/current/api/pymongo/cursor.html) object. These can also have operations performed on them such as `count` to get the amount of records or [`distinct(distinct_field)` (https://docs.mongodb.com/manual/reference/method/db.collection.distinct/ (https://docs.mongodb.com/manual/reference/method/db.collection.distinct/)) to get unique records according to that particular field.

The useful part of a `Cursor`, however, is that it can be iterated over like a Python `list`. Each item in the cursor is an object from which fields can be accessed. For example, to get the `RatingValue` of each establishment in Swansea, the following code would be used:

In [16]:

```
# You don't need to write anything here
for c in db.swansea.find({'RatingValue': 5}):
    print(c['RatingValue'])
    # We don't want to print out all of them so break out of the loop now
    break
```

```
5
```

In [17]:

```
 #You don't need to write anything here
for c in db.swansea.find():
    print(db.swansea.find_one({'RatingValue': 5}))
    # We don't want to print out all of them so break out of the loop now
    break
```

```
{'RatingKey': 'fhrs_5_en-GB', 'RatingValue': 5, 'PostCode': 'SA2 0AY',
'BusinessName': '360 Beach and Watersports Centre', 'Scores': {'Hygien
e': 0, 'ConfidenceInManagement': 0, 'Structural': 5}, 'BusinessType':
'Restaurant/Cafe/Canteen', 'AddressLine4': 'Swansea', 'BusinessTypeI
D': '1', 'Region': 'wales', 'RatingDate': datetime.datetime(2016, 1,
6, 0, 0), 'LocalAuthorityName': 'Swansea', 'Geocode': None, 'LocalAuth
orityCode': '568', 'LocalAuthorityEmailAddress': 'FoodandSafety@swanse
a.gov.uk', 'LocalAuthorityBusinessID': '152289', 'FHRSID': '492474',
'_id': ObjectId('5be545edc4cc3a0001c72e6c'), 'Structural': 5, 'SchemeT
ype': 'FHRS', 'LocalAuthorityWebSite': 'http://www.swansea.gov.uk', 'H
ygiene': 0, 'AddressLine2': 'Mumbles Road', 'ConfidenceInManagement':
0, 'AddressLine3': 'Brynmill', 'NewRatingPending': 'False'}
```

Write a query which gets each different type of business in the Southampton collection.

In [21]:

```
# YOUR CODE HERE
db.southampton.distinct('BusinessType')
```

Out[21]:

```
['Restaurant/Cafe/Canteen',
 'Retailers - other',
 'Hotel/bed & breakfast/guest house',
 'Hospitals/Childcare/Caring Premises',
 'Other catering premises',
 'Retailers - supermarkets/hypermarkets',
 'Mobile caterer',
 'Takeaway/sandwich shop',
 'Pub/bar/nightclub',
 'School/college/university',
 'Manufacturers/packers']
```

MongoDB Aggregation Framework

For performing SQL `GROUP BY` operations such as `MIN` or `MAX` on objects, the MongoDB Aggregation framework is what you'll need to use. It is more complicated than the simple `find` queries, as it has a "pipeline" of different operations. For our purposes, the one we wish to concentrate on is the `$group` pipeline.

To use it, we call `db.collection.aggregate`, and pass a list as the first parameter. Within the list, there are a series of `dict` objects representing a stage in the pipeline as `{"$stage": {"key": "value} }"`.

For grouping then, we would have key `"$group" with a value of a dict.  In the dict, we have the pairs`"output_field": {"$operator": "field_name"}`

A simple example can be seen below, which gives the sum of each different business type in York. Note the following things about it:

- The list parameter, with the nested objects inside it.
- The `_id` of $BusinessType - this is the field we're grouping on. In this case, the `$` sign means that we are getting the value of the field.
- The output field `count` has the `"$sum"`, with each instance being given a weighting of 1. To double the value of this field, we could simply use `{"$sum": 2}` instead.

In [28]:

```
# You don't need to write anything here
coll = db.york.aggregate(
    [
        {"$group": { "_id": "$BusinessType" , "count": {"$sum": 1} } }
    ]
)
for dot in coll:
    print(dot)
```

```
{'count': 50, '_id': 'Mobile caterer'}
{'count': 53, '_id': 'Retailers - supermarkets/hypermarkets'}
{'count': 15, '_id': 'Distributors/Transporters'}
{'count': 1, '_id': 'Importers/Exporters'}
{'count': 183, '_id': 'Takeaway/sandwich shop'}
{'count': 10, '_id': 'Farmers/growers'}
{'count': 273, '_id': 'Other catering premises'}
{'count': 340, '_id': 'Retailers - other'}
{'count': 93, '_id': 'School/college/university'}
{'count': 184, '_id': 'Hotel/bed & breakfast/guest house'}
{'count': 25, '_id': 'Manufacturers/packers'}
{'count': 232, '_id': 'Pub/bar/nightclub'}
{'count': 144, '_id': 'Hospitals/Childcare/Caring Premises'}
{'count': 432, '_id': 'Restaurant/Cafe/Canteen'}
```

Write a function which gives a count of the different `RatingValue` in `db.uk`.

In [23]:

```
# YOUR CODE HERE
coll = db.uk.aggregate(
    [
        {"$group": { "_id": "$RatingValue", "count": {"$sum": 1} } }
    ]
)
for dot in coll:
    print(dot)
```

```
{'count': 1411, '_id': 0}
{'count': 88363, '_id': None}
{'count': 11096, '_id': 2}
{'count': 12831, '_id': 1}
{'count': 85219, '_id': 4}
{'count': 270611, '_id': 5}
{'count': 42288, '_id': 3}
```

In [ ]:

```

```