J. Strait PyMongo

April 28, 2020

1 PyMongo!

1.1 Connect to the Cluster!

In this activity, you will use Python to interact with a Mongo Database stored on a remote (Cloud) server! This is very realistic in terms of how you might interact with a NoSQL database like MongoDB. It will be large and not stored on a single machine and certainly not on your local machine. We have a loaded a collection of documents about restaurants in New York City into a MongoDB cluster. First you will import the necessary modules, PyMongo and Pandas. Next you establish a connection to the cluster (either Morgan's or Dr. Rigas') using a connection string. Lastly you specify the database and collection within the cluster that we want to work with.

[3]:
```
import pymongo
from pymongo import MongoClient
import pandas as pd
client = MongoClient('mongodb://DS220Users:ds220fa19@msterling-shard-00-00-0pfrg.
 ↪mongodb.net:27017,msterling-shard-00-01-0pfrg.mongodb.net:
 ↪27017,msterling-shard-00-02-0pfrg.mongodb.net:27017/test?
 ↪ssl=true&replicaSet=MSterling-shard-0&authSource=admin&w=majority')  #Use one␣
 ↪of the instructors' connection strings for the Restaurant database
db = client.test
restaurants = db.restaurants
```

1.2 Let's Query!

Querying is basically the same as you would in your command prompt. The only difference is that PyMongo doesn't support .pretty(). For a simple example: to find one of the Bronx restaurants run the cell below and see the output.

[4]:
```
print(restaurants.find_one({"borough":"Bronx"}))
```

```
{'grades': [{'date': datetime.datetime(2014, 3, 3, 0, 0), 'grade': 'A', 'score':
10}, {'date': datetime.datetime(2013, 9, 26, 0, 0), 'grade': 'A', 'score': 10},
{'date': datetime.datetime(2013, 3, 19, 0, 0), 'grade': 'A', 'score': 10},
{'date': datetime.datetime(2012, 8, 29, 0, 0), 'grade': 'A', 'score': 11},
{'date': datetime.datetime(2011, 8, 17, 0, 0), 'grade': 'A', 'score': 13}],
'borough': 'Bronx', '_id': ObjectId('5cdf44b7905f6d5ca0bcbf0b'), 'address':
{'building': '277', 'coord': [-73.8941893, 40.8634684], 'zipcode': '10458',
```

```
'street': 'East Kingsbridge Road'}, 'restaurant_id': '40364296', 'cuisine':
'Chinese', 'name': 'Happy Garden'}
```

Now if we wanted to find ALL of the Bronx restaurants, we simply cannot print the query. It will create a cursor object. To see all of the query results, we need to use a for loop. See below. This example is limited to output the first 5 documents to save space, but if you wanted to see all of the restaurants, just remove the .limit(5).

[5]:
```
for restaurant in restaurants.find({"borough":"Bronx"}).limit(5):
    print(restaurant)
```

```
{'grades': [{'date': datetime.datetime(2014, 3, 3, 0, 0), 'grade': 'A', 'score':
10}, {'date': datetime.datetime(2013, 9, 26, 0, 0), 'grade': 'A', 'score': 10},
{'date': datetime.datetime(2013, 3, 19, 0, 0), 'grade': 'A', 'score': 10},
{'date': datetime.datetime(2012, 8, 29, 0, 0), 'grade': 'A', 'score': 11},
{'date': datetime.datetime(2011, 8, 17, 0, 0), 'grade': 'A', 'score': 13}],
'borough': 'Bronx', '_id': ObjectId('5cdf44b7905f6d5ca0bcbf0b'), 'address':
{'building': '277', 'coord': [-73.8941893, 40.8634684], 'zipcode': '10458',
'street': 'East Kingsbridge Road'}, 'restaurant_id': '40364296', 'cuisine':
'Chinese', 'name': 'Happy Garden'}
{'grades': [{'date': datetime.datetime(2014, 6, 21, 0, 0), 'grade': 'A',
'score': 5}, {'date': datetime.datetime(2012, 7, 11, 0, 0), 'grade': 'A',
'score': 10}], 'borough': 'Bronx', '_id': ObjectId('5cdf44b7905f6d5ca0bcbf0f'),
'address': {'building': '658', 'coord': [-73.81363999999999, 40.82941100000001],
'zipcode': '10465', 'street': 'Clarence Ave'}, 'restaurant_id': '40364363',
'cuisine': 'American', 'name': 'Manhem Club'}
{'grades': [{'date': datetime.datetime(2014, 5, 28, 0, 0), 'grade': 'A',
'score': 11}, {'date': datetime.datetime(2013, 6, 19, 0, 0), 'grade': 'A',
'score': 4}, {'date': datetime.datetime(2012, 6, 15, 0, 0), 'grade': 'A',
'score': 3}], 'borough': 'Bronx', '_id': ObjectId('5cdf44b7905f6d5ca0bcbed8'),
'address': {'building': '2300', 'coord': [-73.8786113, 40.8502883], 'zipcode':
'10460', 'street': 'Southern Boulevard'}, 'restaurant_id': '40357217',
'cuisine': 'American', 'name': 'Wild Asia'}
{'grades': [{'date': datetime.datetime(2013, 12, 30, 0, 0), 'grade': 'A',
'score': 8}, {'date': datetime.datetime(2013, 1, 8, 0, 0), 'grade': 'A',
'score': 10}, {'date': datetime.datetime(2012, 6, 12, 0, 0), 'grade': 'B',
'score': 15}], 'borough': 'Bronx', '_id': ObjectId('5cdf44b7905f6d5ca0bcbef1'),
'address': {'building': '1236', 'coord': [-73.8893654, 40.81376179999999],
'zipcode': '10474', 'street': '238 Spofford Ave'}, 'restaurant_id': '40363289',
'cuisine': 'Chinese', 'name': 'Happy Garden'}
{'grades': [{'date': datetime.datetime(2014, 3, 3, 0, 0), 'grade': 'A', 'score':
2}, {'date': datetime.datetime(2013, 9, 11, 0, 0), 'grade': 'A', 'score': 6},
{'date': datetime.datetime(2013, 1, 24, 0, 0), 'grade': 'A', 'score': 10},
{'date': datetime.datetime(2011, 11, 23, 0, 0), 'grade': 'A', 'score': 9},
{'date': datetime.datetime(2011, 3, 10, 0, 0), 'grade': 'B', 'score': 14}],
'borough': 'Bronx', '_id': ObjectId('5cdf44b7905f6d5ca0bcbed1'), 'address':
{'building': '1007', 'coord': [-73.856077, 40.848447], 'zipcode': '10462',
'street': 'Morris Park Ave'}, 'restaurant_id': '30075445', 'cuisine': 'Bakery',
'name': 'Morris Park Bake Shop'}
```

This is not the prettiest to look at. So, we can make use of pandas dataframes! First we establish an empty list. We loop through the restaurants in the query and append them to the list. Finally, we make a dataframe out of the list of json objects and display our result. Run the cell below to see a dataframe of 15 of the Indian cuisine restaurants in Brooklyn in our database. YOU MAY NEED TO ADD SOME CODE TO DISPLAY THE DATAFRAME.

[7]:
```
brooklynIndian = []
for rest in restaurants.find({"$and": [{"cuisine":"Indian"}, {"borough":␣
 ↪"Brooklyn"}]}).limit(15):
    brooklynIndian.append(rest)
testDF = pd.DataFrame(brooklynIndian)
testDF
```

[7]:
```
                         _id  \
0   5cdf44b7905f6d5ca0bcc10f
1   5cdf44b9905f6d5ca0bccb03
2   5cdf44b9905f6d5ca0bcceaa
3   5cdf44b9905f6d5ca0bccf70
4   5cdf44b9905f6d5ca0bcd38e
5   5cdf44ba905f6d5ca0bcd9e8
6   5cdf44ba905f6d5ca0bcdbe8
7   5cdf44bb905f6d5ca0bce14a
8   5cdf44bb905f6d5ca0bce41e
9   5cdf44bb905f6d5ca0bce7df
10  5cdf44bb905f6d5ca0bce87a
11  5cdf44bb905f6d5ca0bce856
12  5cdf44bb905f6d5ca0bce963
13  5cdf44bb905f6d5ca0bceb11
14  5cdf44bc905f6d5ca0bceeb3

                                              address   borough cuisine  \
0   {'building': '7407', 'coord': [-74.0272951, 40...  Brooklyn  Indian
1   {'building': '7315', 'coord': [-74.0271258, 40...  Brooklyn  Indian
2   {'building': '301', 'coord': [-73.973266, 40.6...  Brooklyn  Indian
3   {'building': '142', 'coord': [-73.990505100000...  Brooklyn  Indian
4   {'building': '473', 'coord': [-73.9869382, 40...  Brooklyn  Indian
5   {'building': '368', 'coord': [-73.971424, 40.6...  Brooklyn  Indian
6   {'building': '678', 'coord': [-73.956518, 40.6...  Brooklyn  Indian
7   {'building': '563', 'coord': [-73.969748900000...  Brooklyn  Indian
8   {'building': '310', 'coord': [-73.9831495, 40...  Brooklyn  Indian
9   {'building': '1215', 'coord': [-73.864654, 40...  Brooklyn  Indian
10  {'building': '600', 'coord': [-73.960807, 40.6...  Brooklyn  Indian
11  {'building': '1203', 'coord': [-73.8650067, 40...  Brooklyn  Indian
12  {'building': '513', 'coord': [-73.950649, 40.7...  Brooklyn  Indian
13  {'building': '151', 'coord': [-73.99219, 40.69...  Brooklyn  Indian
14  {'building': '595', 'coord': [-73.963866799999...  Brooklyn  Indian

                                                grades  \
```

```
0   [{'date': 2014-12-17 00:00:00, 'grade': 'Z', '...
1   [{'date': 2014-12-10 00:00:00, 'grade': 'A', '...
2   [{'date': 2014-09-22 00:00:00, 'grade': 'A', '...
3   [{'date': 2014-12-10 00:00:00, 'grade': 'A', '...
4   [{'date': 2014-08-27 00:00:00, 'grade': 'A', '...
5   [{'date': 2014-06-03 00:00:00, 'grade': 'A', '...
6   [{'date': 2014-06-19 00:00:00, 'grade': 'B', '...
7   [{'date': 2014-07-22 00:00:00, 'grade': 'A', '...
8   [{'date': 2014-09-04 00:00:00, 'grade': 'B', '...
9   [{'date': 2014-09-30 00:00:00, 'grade': 'B', '...
10  [{'date': 2014-12-18 00:00:00, 'grade': 'A', '...
11  [{'date': 2014-08-21 00:00:00, 'grade': 'C', '...
12  [{'date': 2014-11-19 00:00:00, 'grade': 'C', '...
13  [{'date': 2014-07-10 00:00:00, 'grade': 'A', '...
14  [{'date': 2014-12-10 00:00:00, 'grade': 'A', '...
```

	name	restaurant_id
0	India Passage Restaurant	40384479
1	Taj Mahal Indian Restaurant	40756209
2	Joy Indian Restaurant	40928416
3	Britain Indian Restaurant	40944839
4	Kinara Indian Restaurant	41060552
5	Kinaras Indian Food	41217162
6	Bombay Masala	41247331
7	Madina Restaurant	41335328
8	Baluchi'S	41377014
9	Motin Spicy & Sweets	41418911
10	King Of Tandoor	41427508
11	Bismillah Kabab & Curry	41425640
12	Curry Heaven	41440204
13	Curry Heights	41459557
14	Sapid Indian Restaurant	41499963

1.3 Your turn!

Use the example above to complete the following exercises. For documentation on Mongo querying visit https://docs.mongodb.com/manual/tutorial/query-documents/ and https://docs.mongodb.com/manual/tutorial/query-embedded-documents/

2 1. In the cell below, fill in the missing details to display the fields restaurant_id, name, borough and cuisine, but exclude the ID field for the first 10 documents in the database.

[8]:
```
for rest in restaurants.find({},{"restaurant_id": 1,"name":1,"borough": 1,␣
↪"cuisine": 1, "_id":0}).limit(10):
  print(rest)
```

```
{'restaurant_id': '40356018', 'name': 'Riviera Caterer', 'cuisine': 'American',
'borough': 'Brooklyn'}
{'restaurant_id': '40361618', 'name': "Sal'S Deli", 'cuisine': 'Delicatessen',
'borough': 'Queens'}
{'restaurant_id': '40359480', 'name': '1 East 66Th Street Kitchen', 'cuisine':
'American', 'borough': 'Manhattan'}
{'restaurant_id': '40364347', 'name': 'Metropolitan Club', 'cuisine':
'American', 'borough': 'Manhattan'}
{'restaurant_id': '40361521', 'name': 'Glorious Food', 'cuisine': 'American',
'borough': 'Manhattan'}
{'restaurant_id': '40356731', 'name': 'Taste The Tropics Ice Cream', 'cuisine':
'Ice Cream, Gelato, Yogurt, Ices', 'borough': 'Brooklyn'}
{'restaurant_id': '40362274', 'name': 'Angelika Film Center', 'cuisine':
'American', 'borough': 'Manhattan'}
{'restaurant_id': '40363945', 'name': "Domino'S Pizza", 'cuisine': 'Pizza',
'borough': 'Manhattan'}
{'restaurant_id': '40364305', 'name': 'Philadelhia Grille Express', 'cuisine':
'Italian', 'borough': 'Brooklyn'}
{'restaurant_id': '40364373', 'name': 'Isle Of Capri Resturant', 'cuisine':
'Italian', 'borough': 'Manhattan'}
```

3 2. Find the first 10 restaurants who have received a *score less than* 50.

[48]:
```
for rest in restaurants.find({"grades.score":{"$lt":50}}).limit(10):
   print(rest)
```

```
{'grades': [{'date': datetime.datetime(2014, 6, 10, 0, 0), 'grade': 'A',
'score': 5}, {'date': datetime.datetime(2013, 6, 5, 0, 0), 'grade': 'A',
'score': 7}, {'date': datetime.datetime(2012, 4, 13, 0, 0), 'grade': 'A',
'score': 12}, {'date': datetime.datetime(2011, 10, 12, 0, 0), 'grade': 'A',
'score': 12}], 'borough': 'Brooklyn', '_id':
ObjectId('5cdf44b7905f6d5ca0bcbee4'), 'address': {'building': '2780', 'coord':
[-73.98241999999999, 40.579505], 'zipcode': '11224', 'street': 'Stillwell
Avenue'}, 'restaurant_id': '40356018', 'cuisine': 'American', 'name': 'Riviera
Caterer'}
{'grades': [{'date': datetime.datetime(2014, 8, 16, 0, 0), 'grade': 'A',
'score': 12}, {'date': datetime.datetime(2013, 8, 27, 0, 0), 'grade': 'A',
'score': 9}, {'date': datetime.datetime(2012, 9, 20, 0, 0), 'grade': 'A',
```

'score': 7}, {'date': datetime.datetime(2011, 9, 29, 0, 0), 'grade': 'A',
'score': 10}], 'borough': 'Queens', '_id': ObjectId('5cdf44b7905f6d5ca0bcbee2'),
'address': {'building': '129-08', 'coord': [-73.839297, 40.78147], 'zipcode':
'11356', 'street': '20 Avenue'}, 'restaurant_id': '40361618', 'cuisine':
'Delicatessen', 'name': "Sal'S Deli"}
{'grades': [{'date': datetime.datetime(2014, 5, 7, 0, 0), 'grade': 'A', 'score':
3}, {'date': datetime.datetime(2013, 5, 3, 0, 0), 'grade': 'A', 'score': 4},
{'date': datetime.datetime(2012, 4, 30, 0, 0), 'grade': 'A', 'score': 6},
{'date': datetime.datetime(2011, 12, 27, 0, 0), 'grade': 'A', 'score': 0}],
'borough': 'Manhattan', '_id': ObjectId('5cdf44b7905f6d5ca0bcbedb'), 'address':
{'building': '1', 'coord': [-73.96926909999999, 40.7685235], 'zipcode': '10065',
'street': 'East   66 Street'}, 'restaurant_id': '40359480', 'cuisine':
'American', 'name': '1 East 66Th Street Kitchen'}
{'grades': [{'date': datetime.datetime(2014, 10, 16, 0, 0), 'grade': 'B',
'score': 24}, {'date': datetime.datetime(2014, 5, 2, 0, 0), 'grade': 'A',
'score': 4}, {'date': datetime.datetime(2013, 4, 2, 0, 0), 'grade': 'A',
'score': 13}, {'date': datetime.datetime(2012, 10, 19, 0, 0), 'grade': 'A',
'score': 12}, {'date': datetime.datetime(2012, 4, 27, 0, 0), 'grade': 'B',
'score': 17}, {'date': datetime.datetime(2011, 11, 29, 0, 0), 'grade': 'A',
'score': 11}], 'borough': 'Manhattan', '_id':
ObjectId('5cdf44b7905f6d5ca0bcbf08'), 'address': {'building': '1', 'coord':
[-73.97166039999999, 40.764832], 'zipcode': '10022', 'street': 'East   60
Street'}, 'restaurant_id': '40364347', 'cuisine': 'American', 'name':
'Metropolitan Club'}
{'grades': [{'date': datetime.datetime(2014, 9, 2, 0, 0), 'grade': 'A', 'score':
12}, {'date': datetime.datetime(2013, 12, 19, 0, 0), 'grade': 'B', 'score': 16},
{'date': datetime.datetime(2013, 5, 28, 0, 0), 'grade': 'A', 'score': 9},
{'date': datetime.datetime(2012, 12, 7, 0, 0), 'grade': 'A', 'score': 13},
{'date': datetime.datetime(2012, 3, 29, 0, 0), 'grade': 'A', 'score': 11}],
'borough': 'Manhattan', '_id': ObjectId('5cdf44b7905f6d5ca0bcbee0'), 'address':
{'building': '522', 'coord': [-73.95171, 40.767461], 'zipcode': '10021',
'street': 'East   74 Street'}, 'restaurant_id': '40361521', 'cuisine':
'American', 'name': 'Glorious Food'}
{'grades': [{'date': datetime.datetime(2014, 7, 14, 0, 0), 'grade': 'A',
'score': 12}, {'date': datetime.datetime(2013, 7, 10, 0, 0), 'grade': 'A',
'score': 8}, {'date': datetime.datetime(2012, 7, 11, 0, 0), 'grade': 'A',
'score': 5}, {'date': datetime.datetime(2012, 2, 23, 0, 0), 'grade': 'A',
'score': 8}], 'borough': 'Brooklyn', '_id':
ObjectId('5cdf44b7905f6d5ca0bcbed7'), 'address': {'building': '1839', 'coord':
[-73.9482609, 40.6408271], 'zipcode': '11226', 'street': 'Nostrand Avenue'},
'restaurant_id': '40356731', 'cuisine': 'Ice Cream, Gelato, Yogurt, Ices',
'name': 'Taste The Tropics Ice Cream'}
{'grades': [{'date': datetime.datetime(2014, 4, 3, 0, 0), 'grade': 'A', 'score':
9}, {'date': datetime.datetime(2013, 4, 5, 0, 0), 'grade': 'A', 'score': 4},
{'date': datetime.datetime(2012, 3, 21, 0, 0), 'grade': 'A', 'score': 13},
{'date': datetime.datetime(2011, 4, 27, 0, 0), 'grade': 'A', 'score': 5}],
'borough': 'Manhattan', '_id': ObjectId('5cdf44b7905f6d5ca0bcbeea'), 'address':
{'building': '18', 'coord': [-73.996984, 40.72589], 'zipcode': '10012',

```
'street': 'West Houston Street'}, 'restaurant_id': '40362274', 'cuisine':
'American', 'name': 'Angelika Film Center'}
{'grades': [{'date': datetime.datetime(2014, 12, 8, 0, 0), 'grade': 'A',
'score': 13}, {'date': datetime.datetime(2014, 5, 5, 0, 0), 'grade': 'B',
'score': 18}, {'date': datetime.datetime(2013, 4, 5, 0, 0), 'grade': 'A',
'score': 13}, {'date': datetime.datetime(2012, 3, 30, 0, 0), 'grade': 'A',
'score': 9}], 'borough': 'Manhattan', '_id':
ObjectId('5cdf44b7905f6d5ca0bcbf01'), 'address': {'building': '148', 'coord':
[-73.9806854, 40.7778589], 'zipcode': '10023', 'street': 'West   72 Street'},
'restaurant_id': '40363945', 'cuisine': 'Pizza', 'name': "Domino'S Pizza"}
{'grades': [{'date': datetime.datetime(2014, 2, 25, 0, 0), 'grade': 'A',
'score': 12}, {'date': datetime.datetime(2013, 6, 27, 0, 0), 'grade': 'A',
'score': 7}, {'date': datetime.datetime(2012, 12, 3, 0, 0), 'grade': 'A',
'score': 10}, {'date': datetime.datetime(2011, 11, 9, 0, 0), 'grade': 'A',
'score': 12}], 'borough': 'Brooklyn', '_id':
ObjectId('5cdf44b7905f6d5ca0bcbf06'), 'address': {'building': '10004', 'coord':
[-74.03400479999999, 40.6127077], 'zipcode': '11209', 'street': '4 Avenue'},
'restaurant_id': '40364305', 'cuisine': 'Italian', 'name': 'Philadelhia Grille
Express'}
{'grades': [{'date': datetime.datetime(2014, 9, 16, 0, 0), 'grade': 'A',
'score': 13}, {'date': datetime.datetime(2014, 2, 24, 0, 0), 'grade': 'A',
'score': 10}, {'date': datetime.datetime(2013, 5, 3, 0, 0), 'grade': 'A',
'score': 10}, {'date': datetime.datetime(2012, 8, 20, 0, 0), 'grade': 'A',
'score': 7}, {'date': datetime.datetime(2012, 2, 13, 0, 0), 'grade': 'A',
'score': 9}], 'borough': 'Manhattan', '_id':
ObjectId('5cdf44b7905f6d5ca0bcbf0c'), 'address': {'building': '1028', 'coord':
[-73.966032, 40.762832], 'zipcode': '10065', 'street': '3 Avenue'},
'restaurant_id': '40364373', 'cuisine': 'Italian', 'name': 'Isle Of Capri
Resturant'}
```

4 3. Return *only* the name, cuisine, and borough for 20 documents that are *NOT IN* the Bronx, Queens, or Staten Island (Hint: There is a Mongo method called $nin for "not in".)

[30]:
```
for rest in restaurants.find({"borough":{"$nin":["Queens", "Staten Island",␣
↪"Bronx"]}},{"name":1, "cuisine":1, "borough":1, "_id":0}).limit(20):
  print(rest)
```

```
{'name': 'Riviera Caterer', 'cuisine': 'American', 'borough': 'Brooklyn'}
{'name': '1 East 66Th Street Kitchen', 'cuisine': 'American', 'borough':
'Manhattan'}
{'name': 'Metropolitan Club', 'cuisine': 'American', 'borough': 'Manhattan'}
{'name': 'Glorious Food', 'cuisine': 'American', 'borough': 'Manhattan'}
{'name': 'Taste The Tropics Ice Cream', 'cuisine': 'Ice Cream, Gelato, Yogurt,
Ices', 'borough': 'Brooklyn'}
{'name': 'Angelika Film Center', 'cuisine': 'American', 'borough': 'Manhattan'}
```

```
{'name': "Domino'S Pizza", 'cuisine': 'Pizza', 'borough': 'Manhattan'}
{'name': 'Philadelhia Grille Express', 'cuisine': 'Italian', 'borough':
'Brooklyn'}
{'name': 'Isle Of Capri Resturant', 'cuisine': 'Italian', 'borough':
'Manhattan'}
{'name': 'Seuda Foods', 'cuisine': 'Jewish/Kosher', 'borough': 'Brooklyn'}
{'name': 'P & S Deli Grocery', 'cuisine': 'American', 'borough': 'Manhattan'}
{'name': 'Cafe Metro', 'cuisine': 'American', 'borough': 'Manhattan'}
{'name': "Bully'S Deli", 'cuisine': 'Delicatessen', 'borough': 'Manhattan'}
{'name': "Wendy'S", 'cuisine': 'Hamburgers', 'borough': 'Brooklyn'}
{'name': 'The Country Cafe', 'cuisine': 'Turkish', 'borough': 'Manhattan'}
{'name': 'Downtown Deli', 'cuisine': 'American', 'borough': 'Manhattan'}
{'name': "Olive'S", 'cuisine': 'Bakery', 'borough': 'Manhattan'}
{'name': 'Golden Pavillion', 'cuisine': 'Chinese', 'borough': 'Brooklyn'}
{'name': 'Peter Luger Steakhouse', 'cuisine': 'Steak', 'borough': 'Brooklyn'}
{'name': 'Dj Reynolds Pub And Restaurant', 'cuisine': 'Irish', 'borough':
'Manhattan'}
```

5 4. Due to a recent inventory mishap, a client needs the names and addresses for the Mexican restaurants north of the Central Park Zoo (lattitude: 40.7678). Do not return the id. This client will only visit the first 20 restaurants in ascending order by name, so only return those. Hint: will need to make use of pymongo.ASCENDING. You can also use a a property of the address atribute, address.coord. You are lucky in this case that the longitudes are all negative numbers so you can find an easy way to search for a lattitude number greater than the lattitude of interest.

[55]:
```
for rest in restaurants.find({"$and":[{"address.coord":{"$gt":40.7678}},␣
↪{"cuisine":"Mexican"}]},{"name":1, "address":1, "_id":0}).sort("name", pymongo.
↪ASCENDING).limit(20):
    print(rest)
```

```
{'address': {'building': '744', 'coord': [-73.889686, 40.844425], 'zipcode':
'10457', 'street': 'East Tremont Avenue'}, 'name': '3 Mounts'}
{'address': {'building': '414', 'coord': [-73.92245900000002, 40.808622],
'zipcode': '10454', 'street': 'East 138 Street'}, 'name': '414 Latino Restaurant
Sports Bar'}
{'address': {'building': '2445', 'coord': [-73.8989026, 40.8616336], 'zipcode':
'10468', 'street': 'Creston Avenue'}, 'name': 'A&E Tenochtitlan Deli &
Taqueria'}
{'address': {'building': '2888', 'coord': [-73.9653136, 40.8055357], 'zipcode':
'10025', 'street': 'Broadway'}, 'name': 'Amigos'}
{'address': {'building': '783', 'coord': [-73.8645049, 40.85462340000001],
```

'zipcode': '10462', 'street': 'Lyding Ave'}, 'name': 'Azul Tequila Mexican
Restaurant'}
{'address': {'building': '1770', 'coord': [-73.8686874, 40.8396451], 'zipcode':
'10460', 'street': 'East Tremont Avenue'}, 'name': 'Burger One Grill'}
{'address': {'building': '3764', 'coord': [-73.8215929, 40.825758], 'zipcode':
'10465', 'street': 'East Tremont Avenue'}, 'name': 'Cabo Restaurant'}
{'address': {'building': '368', 'coord': [-73.97644199999999, 40.7810076],
'zipcode': '10024', 'street': 'Columbus Avenue'}, 'name': 'Cafe Frida'}
{'address': {'building': '1838', 'coord': [-73.955101, 40.80038500000001],
'zipcode': '10026', 'street': 'Adam Clayton Powell Jr Boulevard'}, 'name':
'Cantina'}
{'address': {'building': '1470', 'coord': [-73.9536119, 40.7705841], 'zipcode':
'10075', 'street': '1 Avenue'}, 'name': 'Canyon Road Grill'}
{'address': {'building': '800', 'coord': [-73.9065883, 40.8125405], 'zipcode':
'10455', 'street': 'East 149 Street'}, 'name': 'Carnitas El Atoradero'}
{'address': {'building': '898', 'coord': [-73.9670733, 40.7989195], 'zipcode':
'10025', 'street': 'Amsterdam Avenue'}, 'name': 'Casa Mexicana'}
{'address': {'building': '2799', 'coord': [-73.96805750000001, 40.8028278],
'zipcode': '10025', 'street': 'Broadway'}, 'name': 'Cascabel Taqueria'}
{'address': {'building': '1556', 'coord': [-73.9539549, 40.774339], 'zipcode':
'10028', 'street': '2 Avenue'}, 'name': 'Cascabel Taqueria'}
{'address': {'building': '2126', 'coord': [-73.940585, 40.792663], 'zipcode':
'10029', 'street': '2 Avenue'}, 'name': 'Cascalote Latin Bistro'}
{'address': {'building': '536', 'coord': [-73.91823099999999, 40.806821],
'zipcode': '10454', 'street': 'East 138 Street'}, 'name': 'Cervantes
Restaurant'}
{'address': {'building': '768', 'coord': [-73.969911, 40.795201], 'zipcode':
'10025', 'street': 'Amsterdam Avenue'}, 'name': 'Chicojulio'}
{'address': {'building': '1221', 'coord': [-92.7182167, 41.7467777], 'zipcode':
'10020', 'street': '6 Avenue'}, 'name': 'Chipotle Mexican Grill'}
{'address': {'building': '350', 'coord': [-93.2069217, 43.14769159999999],
'zipcode': '10018', 'street': '5 Avenue'}, 'name': 'Chipotle Mexican Grill'}
{'address': {'building': '2843', 'coord': [-73.9667671, 40.804585], 'zipcode':
'10025', 'street': 'Broadway'}, 'name': 'Chipotle Mexican Grill'}

6 5. Did you know that MongoDB also supports regular expressions? We have a client that can't remember the name of a restaurant they rated. All this client knows is that it was an American cuisine restaurant in Manhattan whose name started with "Mad". We have a few of these in our database. We will create a dataframe of these restaurants to send to our client. Fill in the missing detail below to run this and print the result (Hint: In Mongo, you can indicate that you want the string to START with a particular pattern by using the '^' as an anchor at the beginning of the string).

[31]:
```
mysteryRestaurant = []
for rest in restaurants.find({"name": {"$regex":"^Mad"}, "cuisine":"American",␣
 ↪"borough": "Manhattan"}):
    mysteryRestaurant.append(rest)
result = pd.DataFrame(mysteryRestaurant)
result
```

[31]:
```
                        _id  \
0  5cdf44b8905f6d5ca0bcc437
1  5cdf44b8905f6d5ca0bcc7cf
2  5cdf44b9905f6d5ca0bccce9
3  5cdf44b9905f6d5ca0bcd6c1
4  5cdf44bb905f6d5ca0bcdf6a
5  5cdf44bb905f6d5ca0bce68a
6  5cdf44bc905f6d5ca0bcf55a
7  5cdf44be905f6d5ca0bd0481
8  5cdf44bf905f6d5ca0bd1b91

                                              address    borough   cuisine  \
0  {'building': '51', 'coord': [-73.9860597, 40.7...  Manhattan  American
1  {'building': '94', 'coord': [-74.000002, 40.72...  Manhattan  American
2  {'building': '1442', 'coord': [-73.95623719999...  Manhattan  American
3  {'building': '360', 'coord': [-73.981973, 40.7...  Manhattan  American
4  {'building': '299', 'coord': [-73.979434000000...  Manhattan  American
5  {'building': '965', 'coord': [-73.9650056, 40...  Manhattan  American
6  {'building': '420', 'coord': [-73.976501, 40.7...  Manhattan  American
7  {'building': '4', 'coord': [-73.99426609999999...  Manhattan  American
8  {'building': '27', 'coord': [-73.9992901999999...  Manhattan  American

                                              grades                    name  \
0  [{'date': 2015-01-13 00:00:00, 'grade': 'A', '...          Madison Square
1  [{'date': 2014-03-14 00:00:00, 'grade': 'A', '...                Madame X
2  [{'date': 2014-12-10 00:00:00, 'grade': 'A', '...  Mad River Bar & Grille
3  [{'date': 2014-05-05 00:00:00, 'grade': 'A', '...       Mad Hatter Saloon
4  [{'date': 2014-09-02 00:00:00, 'grade': 'B', '...          Madison & Vine
```

```
5  [{'date': 2014-08-28 00:00:00, 'grade': 'A', '...      Madison Restaurant
6  [{'date': 2014-11-25 00:00:00, 'grade': 'A', '...            Madison Deli
7  [{'date': 2014-04-02 00:00:00, 'grade': 'A', '...   Madison Club (Bb7184)
8  [{'date': 2014-08-04 00:00:00, 'grade': 'A', '...   Madison Plaza Gourmet

  restaurant_id
0      40402527
1      40611024
2      40859224
3      41149372
4      41305028
5      41401638
6      41576836
7      41694730
8      50011350
```

Keep in mind: The flexibility of Mongo's schema can cause these dataframes to get very messy. Data cleaning will be a crucial step before analysis when using Python on top of MongoDB!