Data Wrangling with MongoDB

Lesson 5: Problem Set

Most Common City name:

```
#!/usr/bin/env python
"""
Use an aggregation query to answer the following question.

What is the most common city name in our cities collection?

Your first attempt probably identified None as the most frequently occurring city name.
What that actually means is that there are a number of cities without a name field at all.
It's strange that such documents would exist in this collection and, depending on your situation,
might actually warrant further cleaning.

To solve this problem the right way, we should really ignore cities that don't have a name specified.
As a hint ask yourself what pipeline operator allows us to simply filter input?
How do we test for the existence of a field?

Please modify only the 'make_pipeline' function so that it creates and returns an aggregation pipeline
that can be passed to the MongoDB aggregate function. As in our examples in this lesson,
the aggregation pipeline should be a list of one or more dictionary objects.
Please review the lesson examples if you are unsure of the syntax.

Your code will be run against a MongoDB instance that we have provided.
If you want to run this code locally on your machine, you have to install MongoDB,
download and insert the dataset.
For instructions related to MongoDB setup and datasets please see Course Materials.

Please note that the dataset you are using here is a smaller version of the cities collection used in
examples in this lesson. If you attempt some of the same queries that we looked at in the lesson
examples, your results may be different.
"""
def get_db(db_name):
    from pymongo import MongoClient
    client = MongoClient('localhost:27017')
    db = client[db_name]
    return db

def make_pipeline():
    # complete the aggregation pipeline
    pipeline = [{"$group" : {"_id" : "$name",
                             "count" : {"$sum" : 1}}},
                {"$match" : {"_id" : {"$ne" : None}}},
                {"$sort" : { "count" : -1}},
```

```
            {"$limit" : 1}]
    return pipeline

def aggregate(db, pipeline):
    result = db.cities.aggregate(pipeline)
    return result

if __name__ == '__main__':
    # The following statements will be used to test your code by the grader.
    # Any modifications to the code past this point will not be reflected by
    # the Test Run.
    db = get_db('examples')
    pipeline = make_pipeline()
    result = aggregate(db, pipeline)
    import pprint
    pprint.pprint(result["result"][0])
    assert len(result["result"]) == 1
    assert result["result"][0] == {'_id': 'Shahpur', 'count': 6}
```

Output:

```
{u'_id': u'Shahpur', u'count': 6}

Length of the result is correct!

You found the correct city.
```

Region Cities:

```
#!/usr/bin/env python
"""
Use an aggregation query to answer the following question.

Which Region in India has the largest number of cities with longitude between 75 and 80?

Please modify only the 'make_pipeline' function so that it creates and returns an aggregation
pipeline that can be passed to the MongoDB aggregate function. As in our examples in this lesson,
the aggregation pipeline should be a list of one or more dictionary objects.
Please review the lesson examples if you are unsure of the syntax.

Your code will be run against a MongoDB instance that we have provided. If you want to run this
code locally on your machine, you have to install MongoDB, download and insert the dataset.
For instructions related to MongoDB setup and datasets please see Course Materials.

Please note that the dataset you are using here is a smaller version of the twitter dataset used in
examples in this lesson. If you attempt some of the same queries that we looked at in the lesson
examples, your results will be different.
"""
```

```
def get_db(db_name):
    from pymongo import MongoClient
    client = MongoClient('localhost:27017')
    db = client[db_name]
    return db

def make_pipeline():
    # complete the aggregation pipeline
    pipeline = [{"$match" : {"country" : "India"}},
                {"$unwind" : "$isPartOf"},
                {"$match" : {"lon" : {"$gte" : 75, "$lte" : 80}}},
                {"$group" : {"_id" : "$isPartOf",
                             "count" : {"$sum" : 1}}},
                {"$sort" : {"count" : -1}},
                {"$limit" : 1}]

    return pipeline

def aggregate(db, pipeline):
    result = db.cities.aggregate(pipeline)
    return result

if __name__ == '__main__':
    # The following statements will be used to test your code by the grader.
    # Any modifications to the code past this point will not be reflected by
    # the Test Run.
    db = get_db('examples')
    pipeline = make_pipeline()
    result = aggregate(db, pipeline)
    import pprint
    pprint.pprint(result["result"][0])
    assert len(result["result"]) == 1
    assert result["result"][0]["_id"] == 'Tamil Nadu'
    assert result["result"][0]["count"] == 424
```

Output:

```
{u'_id': u'Tamil Nadu', u'count': 424}

Length of the result is correct!

You found the correct region.
```

Average Population:

```
#!/usr/bin/env python
"""
Use an aggregation query to answer the following question.

Extrapolating from an earlier exercise in this lesson, find the average regional city population
for all countries in the cities collection. What we are asking here is that you first calculate the
average city population for each region in a country and then calculate the average of all the
regional averages for a country. As a hint, _id fields in group stages need not be single values.
They can also be compound keys (documents composed of multiple fields). You will use the same
aggregation operator in more than one stage in writing this aggregation query. I encourage you to
write it one stage at a time and test after writing each stage.

Please modify only the 'make_pipeline' function so that it creates and returns an aggregation
pipeline that can be passed to the MongoDB aggregate function. As in our examples in this lesson,
the aggregation pipeline should be a list of one or more dictionary objects.
Please review the lesson examples if you are unsure of the syntax.

Your code will be run against a MongoDB instance that we have provided. If you want to run this code
locally on your machine, you have to install MongoDB, download and insert the dataset.
For instructions related to MongoDB setup and datasets please see Course Materials.

Please note that the dataset you are using here is a smaller version of the cities collection used in
examples in this lesson. If you attempt some of the same queries that we looked at in the lesson
examples, your results may be different.
"""
def get_db(db_name):
    from pymongo import MongoClient
    client = MongoClient('localhost:27017')
    db = client[db_name]
    return db

def make_pipeline():
    # complete the aggregation pipeline
    pipeline = [{"$unwind" : "$isPartOf"},
            {"$group" : {"_id" : {"region" : "$isPartOf",
                                  "country" : "$country"},
                         "reg_pop_avg" : {"$avg" : "$population"}}},
            {"$group" : {"_id" : "$_id.country",
                         "avgRegionalPopulation" : {"$avg" : "$reg_pop_avg"}}}]
    return pipeline

def aggregate(db, pipeline):
    result = db.cities.aggregate(pipeline)
    return result

if __name__ == '__main__':
```

```
# The following statements will be used to test your code by the grader.
# Any modifications to the code past this point will not be reflected by
# the Test Run.
db = get_db('examples')
pipeline = make_pipeline()
result = aggregate(db, pipeline)
import pprint
if len(result["result"]) < 150:
    pprint.pprint(result["result"])
else:
    pprint.pprint(result["result"][:100])
for country in result["result"]:
    if country["_id"] == 'Morocco':
        assert country["_id"] == 'Morocco'
        assert country["avgRegionalPopulation"] == 334771.76190476184
assert {'_id': 'Morocco',
        'avgRegionalPopulation': 334771.76190476184} in result["result"]
```

Output:

```
You found the correct average for 'Kuwait'.

You found the correct average for 'Mali'.
```