031-data-wrangling-with-mongodb

May 3, 2022

3.1. Wrangling Data with MongoDB

```
[5]: from pprint import PrettyPrinter

     import pandas as pd
     from IPython.display import VimeoVideo
     from pymongo import MongoClient
```

```
[6]: VimeoVideo("665412094", h="8334dfab2e", width=600)
```

```
[6]: <IPython.lib.display.VimeoVideo at 0x7fa50038c8b0>
```

```
[7]: VimeoVideo("665412135", h="dcff7ab83a", width=600)
```

```
[7]: <IPython.lib.display.VimeoVideo at 0x7fa5000b89a0>
```

Task 3.1.1: Instantiate a `PrettyPrinter`, and assign it to the variable `pp`.

- Construct a `PrettyPrinter` instance in pprint.

```
[8]: pp = PrettyPrinter(indent = 2)
```

1 Prepare Data

1.1 Connect

```
[9]: VimeoVideo("665412155", h="1ca0dd03d0", width=600)
```

```
[9]: <IPython.lib.display.VimeoVideo at 0x7fa4a928b760>
```

Task 3.1.2: Create a client that connects to the database running at `localhost` on port 27017.

- What's a database client?
- What's a database server?
- Create a client object for a MongoDB instance.

```
[10]: client = MongoClient(host = 'localhost', port = 27017)
```

1.2 Explore

[11]:
```
VimeoVideo("665412176", h="6fea7c6346", width=600)
```

[11]:
```
<IPython.lib.display.VimeoVideo at 0x7fa4a914a460>
```

Task 3.1.3: Print a list of the databases available on `client`.

- What's an iterator?
- List the databases of a server using PyMongo.
- Print output using pprint.

[12]:
```
pp.pprint(list(client.list_databases()))
```

```
[ {'empty': False, 'name': 'admin', 'sizeOnDisk': 40960},
  {'empty': False, 'name': 'air-quality', 'sizeOnDisk': 6987776},
  {'empty': False, 'name': 'config', 'sizeOnDisk': 12288},
  {'empty': False, 'name': 'local', 'sizeOnDisk': 73728}]
```

[13]:
```
VimeoVideo("665412216", h="7d4027dc33", width=600)
```

[13]:
```
<IPython.lib.display.VimeoVideo at 0x7fa4a914a820>
```

Task 3.1.4: Assign the `"air-quality"` database to the variable `db`.

- What's a MongoDB database?
- Access a database using PyMongo.

[14]:
```
db = client['air-quality']
```

[15]:
```
VimeoVideo("665412231", h="89c546b00f", width=600)
```

[15]:
```
<IPython.lib.display.VimeoVideo at 0x7fa4a914ae50>
```

Task 3.1.5: Use the `list_collections` method to print a list of the collections available in `db`.

- What's a MongoDB collection?
- List the collections in a database using PyMongo.

[16]:
```
for c in db.list_collections():
    print(c['name'])
```

```
lagos
system.buckets.lagos
nairobi
system.buckets.nairobi
system.views
dar-es-salaam
system.buckets.dar-es-salaam
```

```
[17]: VimeoVideo("665412252", h="bff2abbdc0", width=600)
```

```
[17]: <IPython.lib.display.VimeoVideo at 0x7fa5000a41f0>
```

Task 3.1.6: Assign the `"nairobi"` collection in `db` to the variable name `nairobi`.

- Access a collection in a database using PyMongo.

```
[18]: nairobi = db['nairobi']
```

```
[19]: VimeoVideo("665412270", h="e4a5f5c84b", width=600)
```

```
[19]: <IPython.lib.display.VimeoVideo at 0x7fa4a80dd520>
```

Task 3.1.7: Use the `count_documents` method to see how many documents are in the `nairobi` collection.

- What's a MongoDB document?
- Count the documents in a collection using PyMongo.

```
[20]: nairobi.count_documents({})
```

```
[20]: 202212
```

```
[21]: VimeoVideo("665412279", h="c2315f3be1", width=600)
```

```
[21]: <IPython.lib.display.VimeoVideo at 0x7fa4a80ddbe0>
```

Task 3.1.8: Use the `find_one` method to retrieve one document from the `nairobi` collection, and assign it to the variable name `result`.

- What's metadata?
- What's semi-structured data?
- Retrieve a document from a collection using PyMongo.

```
[22]: result = nairobi.find_one({})
      pp.pprint(result)
```

```
{ 'P1': 39.67,
  '_id': ObjectId('6261a046e76424a61615daaf'),
  'metadata': { 'lat': -1.3,
                'lon': 36.785,
                'measurement': 'P1',
                'sensor_id': 57,
                'sensor_type': 'SDS011',
                'site': 29},
  'timestamp': datetime.datetime(2018, 9, 1, 0, 0, 2, 472000)}
```

```
[23]: VimeoVideo("665412306", h="e1e913dfd1", width=600)
```

```
[23]: <IPython.lib.display.VimeoVideo at 0x7fa4a80ddca0>
```

Task 3.1.9: Use the `distinct` method to determine how many sensor sites are included in the `nairobi` collection.

- Get a list of distinct values for a key among all documents using PyMongo.

```
[24]: nairobi.distinct('metadata.site')
```

```
[24]: [6, 29]
```

```
[25]: VimeoVideo("665412322", h="4776c6d548", width=600)
```

```
[25]: <IPython.lib.display.VimeoVideo at 0x7fa4a80e8640>
```

Task 3.1.10: Use the `count_documents` method to determine how many readings there are for each site in the `nairobi` collection.

- Count the documents in a collection using PyMongo.

```
[26]: print("Documents from site 6:", nairobi.count_documents({'metadata.site': 6}))
      print("Documents from site 29:", nairobi.count_documents({'metadata.site': 29}))
```

```
Documents from site 6: 70360
Documents from site 29: 131852
```

```
[27]: VimeoVideo("665412344", h="d2354584cd", width=600)
```

```
[27]: <IPython.lib.display.VimeoVideo at 0x7fa4a80e8b80>
```

Task 3.1.11: Use the `aggregate` method to determine how many readings there are for each site in the `nairobi` collection.

- Perform aggregation calculations on documents using PyMongo.

```
[29]: result = nairobi.aggregate(
          [
              {'$group': {'_id': '$metadata.site', 'count': {'$count': {}}}}
          ]
      )
      pp.pprint(list(result))
```

```
[{'_id': 6, 'count': 70360}, {'_id': 29, 'count': 131852}]
```

```
[30]: VimeoVideo("665412372", h="565122c9cc", width=600)
```

```
[30]: <IPython.lib.display.VimeoVideo at 0x7fa4a914a310>
```

Task 3.1.12: Use the `distinct` method to determine how many types of measurements have been taken in the `nairobi` collection.

- Get a list of distinct values for a key among all documents using PyMongo.

```
[31]: nairobi.distinct('metadata.measurement')
```

```
[31]: ['humidity', 'P2', 'temperature', 'P1']
```

```
[32]: VimeoVideo("665412380", h="f7f7a39bb3", width=600)
```

```
[32]: <IPython.lib.display.VimeoVideo at 0x7fa4a914afa0>
```

Task 3.1.13: Use the `find` method to retrieve the PM 2.5 readings from all sites. Be sure to limit your results to 3 records only.

- Query a collection using PyMongo.

```
[34]: result = nairobi.find({'metadata.measurement': 'P2'}).limit(3)
      pp.pprint(list(result))
```

```
[ { 'P2': 34.43,
    '_id': ObjectId('6261a046e76424a616165b3a'),
    'metadata': { 'lat': -1.3,
                  'lon': 36.785,
                  'measurement': 'P2',
                  'sensor_id': 57,
                  'sensor_type': 'SDS011',
                  'site': 29},
    'timestamp': datetime.datetime(2018, 9, 1, 0, 0, 2, 472000)},
  { 'P2': 30.53,
    '_id': ObjectId('6261a046e76424a616165b3b'),
    'metadata': { 'lat': -1.3,
                  'lon': 36.785,
                  'measurement': 'P2',
                  'sensor_id': 57,
                  'sensor_type': 'SDS011',
                  'site': 29},
    'timestamp': datetime.datetime(2018, 9, 1, 0, 5, 3, 941000)},
  { 'P2': 22.8,
    '_id': ObjectId('6261a046e76424a616165b3c'),
    'metadata': { 'lat': -1.3,
                  'lon': 36.785,
                  'measurement': 'P2',
                  'sensor_id': 57,
                  'sensor_type': 'SDS011',
                  'site': 29},
    'timestamp': datetime.datetime(2018, 9, 1, 0, 10, 4, 374000)}]
```

```
[35]: VimeoVideo("665412389", h="8976ea3090", width=600)
```

```
[35]: <IPython.lib.display.VimeoVideo at 0x7fa4a80e8490>
```

Task 3.1.14: Use the `aggregate` method to calculate how many readings there are for each type (`"humidity"`, `"temperature"`, `"P2"`, and `"P1"`) in site `6`.

- Perform aggregation calculations on documents using PyMongo.

[37]:
```
result = nairobi.aggregate(
    [
        {'$match': {'metadata.site': 6}},
        {'$group': {'_id': '$metadata.measurement', 'count': {'$count': {}}}}
    ]
)
pp.pprint(list(result))
```

```
[ {'_id': 'humidity', 'count': 17011},
  {'_id': 'P2', 'count': 18169},
  {'_id': 'temperature', 'count': 17011},
  {'_id': 'P1', 'count': 18169}]
```

[38]:
```
VimeoVideo("665412418", h="0c4b125254", width=600)
```

[38]:
```
<IPython.lib.display.VimeoVideo at 0x7fa4a80e8340>
```

Task 3.1.15: Use the `aggregate` method to calculate how many readings there are for each type (`"humidity"`, `"temperature"`, `"P2"`, and `"P1"`) in site `29`.

- Perform aggregation calculations on documents using PyMongo.

[40]:
```
result = nairobi.aggregate(
    [
        {'$match': {'metadata.site': 29}},
        {'$group': {'_id': '$metadata.measurement', 'count': {'$count': {}}}}
    ]
)
pp.pprint(list(result))
```

```
[ {'_id': 'temperature', 'count': 33019},
  {'_id': 'P1', 'count': 32907},
  {'_id': 'humidity', 'count': 33019},
  {'_id': 'P2', 'count': 32907}]
```

1.3 Import

[41]:
```
VimeoVideo("665412437", h="7a436c7e7e", width=600)
```

[41]:
```
<IPython.lib.display.VimeoVideo at 0x7fa4a80e84f0>
```

Task 3.1.16: Use the `find` method to retrieve the PM 2.5 readings from site `29`. Be sure to limit your results to 3 records only. Since we won't need the metadata for our model, use the `projection` argument to limit the results to the `"P2"` and `"timestamp"` keys only.

- Query a collection using PyMongo.

```
[49]: result = nairobi.find(
          {'metadata.site': 29, 'metadata.measurement': 'P2'},
          projection = {'P2': 1, 'timestamp': 1, '_id': 0}
      )
      pp.pprint(result.next())
```

```
{'P2': 34.43, 'timestamp': datetime.datetime(2018, 9, 1, 0, 0, 2, 472000)}
```

```
[44]: VimeoVideo("665412442", h="494636d1ea", width=600)
```

```
[44]: <IPython.lib.display.VimeoVideo at 0x7fa4a9232400>
```

Task 3.1.17: Read records from your `result` into the DataFrame `df`. Be sure to set the index to `"timestamp"`.

- Create a DataFrame from a dictionary using pandas.

```
[50]: df = pd.DataFrame(result).set_index('timestamp')
      df.head()
```

```
[50]:                             P2
      timestamp
      2018-09-01 00:05:03.941  30.53
      2018-09-01 00:10:04.374  22.80
      2018-09-01 00:15:04.245  13.30
      2018-09-01 00:20:04.869  16.57
      2018-09-01 00:25:04.659  14.07
```

```
[51]: # Check your work
      assert df.shape[1] == 1, f"`df` should have only one column, not {df.shape[1]}."
      assert df.columns == [
          "P2"
      ], f"The single column in `df` should be `'P2'`, not {df.columns[0]}."
      assert isinstance(df.index, pd.DatetimeIndex), "`df` should have a␣
       ↪`DatetimeIndex`."
```