MongoDB Aggregation

Objectives

- Describe simple aggregation operators that process and compute data such as $sort, $limit, $group, $sum, $min, $max, and $avg
- Combine operators to create multi-stage aggregation pipelines
- Build aggregation pipelines that draw insights about the data by returning aggregated values

Set up Environment

- Start mongo server, connect/access to mongo db with provided server creds
- Lists dbs, create/use training database, insert data into collection, show collections

```
theia@theiadocker-craigtrupp8:/home/project$ start_mongo
Starting your mongodb database....
This process can take up to a minute.

Mongodb started, waiting for all services to be ready....

Your mongodb server is now ready to use and available with username: root
password: MjY0My1jcmFpZ3Ry

You can access your mongodb database via:
 • The browser at:
https://craigtrupp8-8081.theiadocker-2-labs-prod-theiak8s-4-tor01.proxy.cog
nitiveclass.ai
 • CommandLine: mongo -u root -p MjY0My1jcmFpZ3Ry --authenticationDatabase
admin local
theia@theiadocker-craigtrupp8:/home/project$ mongo -u root -p
MjY0My1jcmFpZ3Ry --authenticationDatabase admin local
MongoDB shell version v3.6.3
connecting to: mongodb://127.0.0.1:27017/local
MongoDB server version: 3.6.3
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
        http://docs.mongodb.org/
```

```
Questions? Try the support group
        http://groups.google.com/group/mongodb-user
Server has startup warnings:
2023-10-15T15:51:49.789+0000 I STORAGE  [initandlisten]
2023-10-15T15:51:49.789+0000 I STORAGE  [initandlisten] ** WARNING: Using
the XFS filesystem is strongly recommended with the WiredTiger storage
engine
2023-10-15T15:51:49.789+0000 I STORAGE  [initandlisten] **          See
http://dochub.mongodb.org/core/prodnotes-filesystem
2023-10-15T15:51:50.696+0000 I CONTROL  [initandlisten]
2023-10-15T15:51:50.696+0000 I CONTROL  [initandlisten] ** WARNING: You are
running on a NUMA machine.
2023-10-15T15:51:50.696+0000 I CONTROL  [initandlisten] **          We
suggest launching mongod like this to avoid performance problems:
2023-10-15T15:51:50.696+0000 I CONTROL  [initandlisten] **
numactl --interleave=all mongod [other options]
2023-10-15T15:51:50.696+0000 I CONTROL  [initandlisten]
> show dbs
admin  0.000GB
local  0.000GB
> use training
switched to db training
> db
training
> use training
switched to db training
> db.marks.insert({"name":"Ramesh","subject":"maths","marks":87})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Ramesh","subject":"english","marks":59})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Ramesh","subject":"science","marks":77})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Rav","subject":"maths","marks":62})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Rav","subject":"english","marks":83})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Rav","subject":"science","marks":71})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Alison","subject":"maths","marks":84})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Alison","subject":"english","marks":82})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Alison","subject":"science","marks":86})
```

```
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Steve","subject":"maths","marks":81})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Steve","subject":"english","marks":89})
WriteResult({ "nInserted" : 1 })
> db.marks.insert({"name":"Steve","subject":"science","marks":77})
WriteResult({ "nInserted" : 1 })
>
db.marks.insert({"name":"Jan","subject":"english","marks":0,"reason":"absen
t"})
WriteResult({ "nInserted" : 1 })
> show collections
marks
```

Exercise 2 - Limiting the rows in the output

- Using the aggregate method and $limit keyword
- Chaining limit method after find (how you discovered yesterday 10/14)

```
> db.marks.aggregate([{"$limit":2}])
{ "_id" : ObjectId("652c0b91826ecc0fa6992376"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992377"), "name" : "Ramesh",
"subject" : "english", "marks" : 59 }
> db.marks.find().limit(2)
{ "_id" : ObjectId("652c0b91826ecc0fa6992376"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992377"), "name" : "Ramesh",
"subject" : "english", "marks" : 59 }
```

Exercise 2A - Find w/Aggregation

- https://www.mongodb.com/docs/manual/reference/method/db.collection.find/#mongodb-method-db.collection.find
- This is a good reference to the documentation to help w/any syntax for mongoDB

```
> db.marks.find({"name":"Rav"}, {name:1, marks:1})
{ "_id" : ObjectId("652d51cf2b1595b85064a10e"), "name" : "Rav", "marks" :
62 }
{ "_id" : ObjectId("652d51cf2b1595b85064a10f"), "name" : "Rav", "marks" :
```

```
83 }
{ "_id" : ObjectId("652d51cf2b1595b85064a110"), "name" : "Rav", "marks" :
71 }
> db.marks.find({"subject":"science"}, {name:1, marks:1}).sort({marks:1})
{ "_id" : ObjectId("652d51cf2b1595b85064a110"), "name" : "Rav", "marks" :
71 }
{ "_id" : ObjectId("652d51cf2b1595b85064a10d"), "name" : "Ramesh", "marks"
: 77 }
{ "_id" : ObjectId("652d51cf2b1595b85064a116"), "name" : "Steve", "marks" :
77 }
{ "_id" : ObjectId("652d51cf2b1595b85064a113"), "name" : "Alison", "marks"
: 86 }
> db.marks.find({subject:"science"}, {name:1, marks:1}).sort({marks:-1})
{ "_id" : ObjectId("652d51cf2b1595b85064a113"), "name" : "Alison", "marks"
: 86 }
{ "_id" : ObjectId("652d51cf2b1595b85064a10d"), "name" : "Ramesh", "marks"
: 77 }
{ "_id" : ObjectId("652d51cf2b1595b85064a116"), "name" : "Steve", "marks" :
77 }
{ "_id" : ObjectId("652d51cf2b1595b85064a110"), "name" : "Rav", "marks" :
71 }
```

- First command uses the find and you can pass a column/key with a value then the second argument is a way of boolean projection (or filtering) the return columns with a 1 for true to limit the output in the return
- The next two commands chain a sort call on the find and filter query to sort the return output for all science scores in ascending then descending order

Exercise 3 - Sorting based on a column

- Sort in Ascending and Descending Order
    - Similar to showing columns the 1, 0, -1 are used as arguments from the offset
    - 1 = Ascending
    - -1 = Descending

```
> db.marks.aggregate([{"$sort": {"marks":1}}])
{ "_id" : ObjectId("652c0bae826ecc0fa6992382"), "name" : "Jan", "subject" :
"english", "marks" : 0, "reason" : "absent" }
{ "_id" : ObjectId("652c0b91826ecc0fa6992377"), "name" : "Ramesh",
"subject" : "english", "marks" : 59 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992379"), "name" : "Rav", "subject" :
```

```
"maths", "marks" : 62 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237b"), "name" : "Rav", "subject" :
"science", "marks" : 71 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992378"), "name" : "Ramesh",
"subject" : "science", "marks" : 77 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992381"), "name" : "Steve", "subject"
: "science", "marks" : 77 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237f"), "name" : "Steve", "subject"
: "maths", "marks" : 81 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237d"), "name" : "Alison",
"subject" : "english", "marks" : 82 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237a"), "name" : "Rav", "subject" :
"english", "marks" : 83 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237c"), "name" : "Alison",
"subject" : "maths", "marks" : 84 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237e"), "name" : "Alison",
"subject" : "science", "marks" : 86 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992376"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992380"), "name" : "Steve", "subject"
: "english", "marks" : 89 }
> db.marks.aggregate([{"$sort": {"marks":-1}}])
{ "_id" : ObjectId("652c0b91826ecc0fa6992380"), "name" : "Steve", "subject"
: "english", "marks" : 89 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992376"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237e"), "name" : "Alison",
"subject" : "science", "marks" : 86 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237c"), "name" : "Alison",
"subject" : "maths", "marks" : 84 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237a"), "name" : "Rav", "subject" :
"english", "marks" : 83 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237d"), "name" : "Alison",
"subject" : "english", "marks" : 82 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237f"), "name" : "Steve", "subject"
: "maths", "marks" : 81 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992378"), "name" : "Ramesh",
"subject" : "science", "marks" : 77 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992381"), "name" : "Steve", "subject"
: "science", "marks" : 77 }
{ "_id" : ObjectId("652c0b91826ecc0fa699237b"), "name" : "Rav", "subject" :
"science", "marks" : 71 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992379"), "name" : "Rav", "subject" :
```

```
"maths", "marks" : 62 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992377"), "name" : "Ramesh",
"subject" : "english", "marks" : 59 }
{ "_id" : ObjectId("652c0bae826ecc0fa6992382"), "name" : "Jan", "subject" :
"english", "marks" : 0, "reason" : "absent" }
```

Exercise 4 - Sorting and limiting

Aggregation usually involves using more than one operator.
A **pipeline** consists of one or more operators declared inside an array.
The operators are comma separated.
Mongodb executes the first operator in the pipeline and sends its output to the next operator.

- Just like shell

```
db.marks.aggregate([
{"$sort":{"marks":-1}},
{"$limit":2}
])
```

```
# Exercise 5 - Group by

The operator $group by, along with operators like $sum, $avg, $min, $max,
allows us to perform grouping operations.

This aggregation pipeline prints the average marks across all subjects.

```

```
db.marks.aggregate([
{
    "$group":{
        "_id":"$subject",
        "average":{"$avg":"$marks"}
        }
}
])
```

Sample Questions

- What are the top 2 marks?”
    - **Sort** appears to have a dict type of value as you can sort by more than one value likely (similar to SQL)
    - **Limit** only appears to take an int value but that also tracks with how you limit in SQL

```
> db.marks.aggregate([{"$sort": {"marks": -1}}, {"$limit":2}])
{ "_id" : ObjectId("652c0b91826ecc0fa6992380"), "name" : "Steve", "subject"
: "english", "marks" : 89 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992376"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
```

- What are the top 5 marks
    - I prefer not encapsulating the fields or aggregation commands, doesn’t look like they are needed to index and get the returns we’re after
    - Aggregate is just apparently able to take a json type list of items for the different keywords which can in itself be a type dictionary or simply a number
        - For ex **sort** is a dictionary value type as you can sort by more than one field
        - And **limit** is simply a numeric value passed as you only pass one type value for the command

```
> db.marks.aggregate([{$sort:{marks:-1}},{$limit:5}])
{ "_id" : ObjectId("652d51cf2b1595b85064a115"), "name" : "Steve", "subject"
: "english", "marks" : 89 }
{ "_id" : ObjectId("652d51cf2b1595b85064a10b"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
{ "_id" : ObjectId("652d51cf2b1595b85064a113"), "name" : "Alison",
"subject" : "science", "marks" : 86 }
{ "_id" : ObjectId("652d51cf2b1595b85064a111"), "name" : "Alison",
"subject" : "maths", "marks" : 84 }
{ "_id" : ObjectId("652d51cf2b1595b85064a10f"), "name" : "Rav", "subject" :
"english", "marks" : 83 }
```

Exercise 5 - GroupBy - Look Closely at Aggregates!

- What is the average mark for each subject

```
> db.marks.find().limit(3)
{ "_id" : ObjectId("652c0b91826ecc0fa6992376"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
```

```
{ "_id" : ObjectId("652c0b91826ecc0fa6992377"), "name" : "Ramesh",
"subject" : "english", "marks" : 59 }
{ "_id" : ObjectId("652c0b91826ecc0fa6992378"), "name" : "Ramesh",
"subject" : "science", "marks" : 77 }
> db.marks.aggregate([
... "$group" : {}
... ^C
```

- ```
-
- # Exercise 5 - Group by
-
- The operator $group by, along with operators like $sum, $avg, $min, $max, allows us to perform grouping operations.
-
- This aggregation pipeline prints the average marks across all subjects.
-
- ```

```
> db.marks.aggregate([
... {
...     "$group":{
...         "_id":"$subject",
...         "average":{"$avg":"$marks"}
...         }
... }
... ])
{ "_id" : "english", "average" : 62.6 }
{ "_id" : "science", "average" : 77.75 }
{ "_id" : "maths", "average" : 78.5 }
```

- This translates into SQL too

```
SELECT subject, AVG(marks)
FROM marks
GROUP BY subject;
```

- Let’s try one level deeper and group by the student, subject and their max/min mark score, and total count of tests per subject
- https://www.statology.org/mongodb-group-by-multiple-fields/

```
> db.marks.aggregate([
... {
... $group : {
... _id : {name:"$name", subject:"$subject"},
... student_subject_score_count : {$sum:1},
... st_sub_min_score : {$min:"$marks"},
... st_sub_max_score : {$max:"$marks"}
... }
... }])
{ "_id" : { "name" : "Jan", "subject" : "english" },
"student_subject_score_count" : 1, "st_sub_min_score" : 0,
"st_sub_max_score" : 0 }
{ "_id" : { "name" : "Steve", "subject" : "science" },
"student_subject_score_count" : 1, "st_sub_min_score" : 77,
"st_sub_max_score" : 77 }
{ "_id" : { "name" : "Steve", "subject" : "english" },
"student_subject_score_count" : 1, "st_sub_min_score" : 89,
"st_sub_max_score" : 89 }
{ "_id" : { "name" : "Rav", "subject" : "maths" },
"student_subject_score_count" : 1, "st_sub_min_score" : 62,
"st_sub_max_score" : 62 }
{ "_id" : { "name" : "Steve", "subject" : "maths" },
"student_subject_score_count" : 1, "st_sub_min_score" : 81,
"st_sub_max_score" : 81 }
{ "_id" : { "name" : "Ramesh", "subject" : "english" },
"student_subject_score_count" : 1, "st_sub_min_score" : 59,
"st_sub_max_score" : 59 }
{ "_id" : { "name" : "Ramesh", "subject" : "maths" },
"student_subject_score_count" : 1, "st_sub_min_score" : 87,
"st_sub_max_score" : 87 }
{ "_id" : { "name" : "Rav", "subject" : "science" },
"student_subject_score_count" : 1, "st_sub_min_score" : 71,
"st_sub_max_score" : 71 }
{ "_id" : { "name" : "Rav", "subject" : "english" },
"student_subject_score_count" : 1, "st_sub_min_score" : 83,
"st_sub_max_score" : 83 }
{ "_id" : { "name" : "Ramesh", "subject" : "science" },
"student_subject_score_count" : 1, "st_sub_min_score" : 77,
```

```
"st_sub_max_score" : 77 }
{ "_id" : { "name" : "Alison", "subject" : "maths" },
"student_subject_score_count" : 1, "st_sub_min_score" : 84,
"st_sub_max_score" : 84 }
{ "_id" : { "name" : "Alison", "subject" : "english" },
"student_subject_score_count" : 1, "st_sub_min_score" : 82,
"st_sub_max_score" : 82 }
{ "_id" : { "name" : "Alison", "subject" : "science" },
"student_subject_score_count" : 1, "st_sub_min_score" : 86,
"st_sub_max_score" : 86 }
```

- So from the student_subject_score_count, we can see there is only a test per subject so let's add some more for Ramesh

```
> db.marks.find().limit(3)
{ "_id" : ObjectId("652d65340ed3fc69af2d5074"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
{ "_id" : ObjectId("652d65340ed3fc69af2d5075"), "name" : "Ramesh",
"subject" : "english", "marks" : 59 }
{ "_id" : ObjectId("652d65340ed3fc69af2d5076"), "name" : "Ramesh",
"subject" : "science", "marks" : 77 }
> more_scores = [{"name":"Ramesh", "subject":"maths", "marks": 82},
{"name":"Ramesh", "subject":"maths", "marks:79}]
2023-10-16T12:38:05.818-0400 E QUERY    [thread1] SyntaxError: unterminated
string literal @(shell):1:103
> more_scores = [{"name":"Ramesh", "subject":"maths", "marks": 82},
{"name":"Ramesh", "subject":"maths", "marks":79}]
[
        {
                "name" : "Ramesh",
                "subject" : "maths",
                "marks" : 82
        },
        {
                "name" : "Ramesh",
                "subject" : "maths",
                "marks" : 79
        }
]
> db.marks.insertMany(more_scores)
{
        "acknowledged" : true,
        "insertedIds" : [
```

```
                ObjectId("652d67220ed3fc69af2d5081"),
                ObjectId("652d67220ed3fc69af2d5082")
        ]
}
> db.marks.find({"$name": "Ramesh", "$subject":"maths"})
Error: error: {
        "ok" : 0,
        "errmsg" : "unknown top level operator: $name",
        "code" : 2,
        "codeName" : "BadValue"
}
> db.marks.find({"name":"Ramesh", "subject":"maths"})
{ "_id" : ObjectId("652d65340ed3fc69af2d5074"), "name" : "Ramesh",
"subject" : "maths", "marks" : 87 }
{ "_id" : ObjectId("652d67220ed3fc69af2d5081"), "name" : "Ramesh",
"subject" : "maths", "marks" : 82 }
{ "_id" : ObjectId("652d67220ed3fc69af2d5082"), "name" : "Ramesh",
"subject" : "maths", "marks" : 79 }
```

- See how the addition of more scores followed by a WHERE type filter on the find shows added scores for Ramesh in the subject math
- Now similar to the above where we aggregated on multiple columns and use a count, min, max score; we'll do the same and use a match argument to just look for Ramesh & Allison
    - https://www.mongodb.com/docs/manual/reference/method/db.collection.aggregate/#mongodb-method-db.collection.aggregate

- WAHOOO - so here is how this breaks down (Multiple Levels)
    - **Match** can use a in argument for multiple values
    - **Group** takes the id as either a single string or dict objects (think multiple groupby values in a sql query
        - Also within the group dict is the column names and aggregate functions to perform

- **Sort** then can take a column created for an aggregate function which to sort by
  - In this example the max score and the highest score or -1

```
> db.marks.aggregate([
... ... {$match: {name:{ $in : ["Ramesh", "Alison"]}}},
... ... {$group: {_id:{name:"$name", subject:"$subject"},
test_count:{$sum:1}, min_score:{$min:"$marks"},
max_score:{$max:"$marks"}}},
... ... {$sort : {max_score : -1}}
... ... ])
{ "_id" : { "name" : "Ramesh", "subject" : "maths" }, "test_count" : 3,
"min_score" : 79, "max_score" : 87 }
{ "_id" : { "name" : "Alison", "subject" : "science" }, "test_count" : 1,
"min_score" : 86, "max_score" : 86 }
{ "_id" : { "name" : "Alison", "subject" : "maths" }, "test_count" : 1,
"min_score" : 84, "max_score" : 84 }
{ "_id" : { "name" : "Alison", "subject" : "english" }, "test_count" : 1,
"min_score" : 82, "max_score" : 82 }
{ "_id" : { "name" : "Ramesh", "subject" : "science" }, "test_count" : 1,
"min_score" : 77, "max_score" : 77 }
{ "_id" : { "name" : "Ramesh", "subject" : "english" }, "test_count" : 1,
"min_score" : 59, "max_score" : 59 }
```

Exercise 6 - Putting it all together

Now let us put together all the operators we have learnt to answer the question. “Who are the top 2 students by average marks?”
This involves:

- finding the average marks per student.
- sorting the output based on average marks in descending order.
- limiting the output to two documents.

```
> db.marks.aggregate([
... {
...     "$group":{
...         "_id":"$name",
...         "average":{"$avg":"$marks"}
```

```
...         }
... },
... {
...     "$sort":{"average":-1}
... },
... {
...     "$limit":2
... }
... ])
{ "_id" : "Alison", "average" : 84 }
{ "_id" : "Steve", "average" : 82.33333333333333 }
```

Practice exercises

1. Find the total marks for each student across all subjects.

```
> db.marks.aggregate([
... {$group: {_id:{name:"$name", subject:"$subject"},
test_count:{$sum:1}}},
... {$sort : {name:-1}}
... ])
{ "_id" : { "name" : "Jan", "subject" : "english" }, "test_count" : 1 }
{ "_id" : { "name" : "Steve", "subject" : "science" }, "test_count" : 1 }
{ "_id" : { "name" : "Steve", "subject" : "english" }, "test_count" : 1 }
{ "_id" : { "name" : "Rav", "subject" : "maths" }, "test_count" : 1 }
{ "_id" : { "name" : "Steve", "subject" : "maths" }, "test_count" : 1 }
{ "_id" : { "name" : "Ramesh", "subject" : "english" }, "test_count" : 1 }
{ "_id" : { "name" : "Ramesh", "subject" : "maths" }, "test_count" : 3 }
{ "_id" : { "name" : "Rav", "subject" : "science" }, "test_count" : 1 }
{ "_id" : { "name" : "Rav", "subject" : "english" }, "test_count" : 1 }
{ "_id" : { "name" : "Ramesh", "subject" : "science" }, "test_count" : 1 }
{ "_id" : { "name" : "Alison", "subject" : "maths" }, "test_count" : 1 }
{ "_id" : { "name" : "Alison", "subject" : "english" }, "test_count" : 1 }
{ "_id" : { "name" : "Alison", "subject" : "science" }, "test_count" : 1 }
```

- I interpreted it differently (I thought total marks meant like count for each subject … here's another take at it

```
> db.marks.aggregate([
...     {
```

```
...         "$group":{"_id":"$name","total":{"$sum":"$marks"}}
...     }
... ])
{ "_id" : "Ramesh", "total" : 384 }
{ "_id" : "Jan", "total" : 0 }
{ "_id" : "Rav", "total" : 216 }
{ "_id" : "Alison", "total" : 252 }
{ "_id" : "Steve", "total" : 247 }
>
```

2. Find the maximum marks scored in each subject.

```
> db.marks.aggregate([
... {$group:{_id:"$subject", max_subject_score:{$max:"$marks"}}},
... {$sort:{max_subject_score:-1}}
... ])
{ "_id" : "english", "max_subject_score" : 89 }
{ "_id" : "maths", "max_subject_score" : 87 }
{ "_id" : "science", "max_subject_score" : 86 }
```

3. Find the minimum & max marks scored by each student.

```
> db.marks.aggregate([
... {$group:{_id:"$name", max_mark:{$max:"$marks"}}},
... {$sort:{max_mark:-1}}
... ])
{ "_id" : "Steve", "max_mark" : 89 }
{ "_id" : "Ramesh", "max_mark" : 87 }
{ "_id" : "Alison", "max_mark" : 86 }
{ "_id" : "Rav", "max_mark" : 83 }
{ "_id" : "Jan", "max_mark" : 0 }
```

```
> db.marks.aggregate([
... {$group:{_id:"$name", min_score:{$min:"$marks"}}},
... {$sort:{min_score:1}}
... ])
{ "_id" : "Jan", "min_score" : 0 }
{ "_id" : "Ramesh", "min_score" : 59 }
{ "_id" : "Rav", "min_score" : 62 }
```

```
{ "_id" : "Steve", "min_score" : 77 }
{ "_id" : "Alison", "min_score" : 82 }
```

4. Find the top two subjects based on average marks
    a. Mine than theirs

```
> db.marks.aggregate([
... ... {$group: {_id:"$subject", subject_avg_mark:{$avg:"$marks"}}},
... ... {$sort: {subject_avg_mark:-1}},
... ... {$limit:2}
... ... ])
{ "_id" : "maths", "subject_avg_mark" : 79.16666666666667 }
{ "_id" : "science", "subject_avg_mark" : 77.75 }
> db.marks.aggregate([
... {
...     "$group":{
...         "_id":"$subject",
...         "average":{"$avg":"$marks"}
...         }
... },
... {
...     "$sort":{"average":-1}
... },
... {
...     "$limit":2
... }
... ])
{ "_id" : "maths", "average" : 79.16666666666667 }
{ "_id" : "science", "average" : 77.75 }
```

- Disconnect from mongodb server

```
> exit
bye
theia@theiadocker-craigtrupp8:/home/project$
```

Alert!!! - How to Start and Delete Mongo Instances which is killing you when trying to spin up a new instance of MongoDB

- https://www.upwork.com/resources/how-to-stop-a-docker-container

```
  theia@theiadocker-craigtrupp8:/home/project$ history 15
    2  mongo -u root -p NDAyNS1jcmFpZ3Ry --authenticationDatabase admin
local
    3  start_mongo
    4  docker container ps -a
    5  docker container rm 312efe63cdbc
    6  docker container ps -a
    7  docker container rm 139f5e9fda4a
    8  docker stop container 139f5e9fda4a
    9  docker stop 139f5e9fda4a
   10  dockr container ps -a
   11  docker container ps -a
   12  docker container rm 139f5e9fda4a
   13  docker container ps -a
   14  clear
   15  history n 15
   16  history 15
```

- Welll … here's the history so you can see the commands
    - docker container ps -a
        - Lists all active docker containers (with ids to stop or delete if needed)
    - docker container rm <id>
        - This is how you delete the container
        - Caveat you can only delete if the container is stopped
    - docker stop <id>
        - This will stop any instance of a docker container that is being spun up

Most importantly … this works if you enter a endless circle of death trying to start up your mongo server

- Here is an active way of deleting docker containers after … i guess exiting from the server again

```
theia@theiadocker-craigtrupp8:/home/project$ docker container ps -a
CONTAINER ID   IMAGE                 COMMAND                  CREATED
STATUS                   PORTS
```

```
NAMES
c90c6859f8e8   mongo-express:0.54.0   "tini -- /docker-ent..."   12 minutes
ago   Up 12 minutes (healthy)   0.0.0.0:8081->8081/tcp, :::8081->8081/tcp
mongo-mongo-express-1
01fb82373bcf   mongo:3.6.3            "docker-entrypoint.s..."   12 minutes
ago   Up 12 minutes (healthy)   0.0.0.0:27017->27017/tcp,
:::27017->27017/tcp   mongo-mongo-1
theia@theiadocker-craigtrupp8:/home/project$ docker container rm
c90c6859f8e8
Error response from daemon: You cannot remove a running container
c90c6859f8e89329d32058f03b536cafe24c454d2855b0d2bdd4f25b916d5e1d. Stop the
container before attempting removal or force remove
theia@theiadocker-craigtrupp8:/home/project$ docker stop c90c6859f8e8
c90c6859f8e8
theia@theiadocker-craigtrupp8:/home/project$ docker stop 01fb82373bcf
01fb82373bcf
theia@theiadocker-craigtrupp8:/home/project$ docker container rm
01fb82373bcf
01fb82373bcf
theia@theiadocker-craigtrupp8:/home/project$ docker container rm
c90c6859f8e8
c90c6859f8e8
theia@theiadocker-craigtrupp8:/home/project$ docker container ps -a
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
theia@theiadocker-craigtrupp8:/home/project$
```