# MongoDB section 4.2 of the project.

4.2 Import the file mongo.json to a collection called docs in a database called proj20DB and write queries to satisfy the following.

- First start the mongoDB community server. (base) $ `brew services start mongodb-community@4.2`

> Successfully started `mongodb-community` (label: homebrew.mxcl.mongodb-commu (base) :~ $

$ `mongo` from another terminal (not sure if this is necessary) to start mongo.

I have the database downloaded into this folder.
*/Users/angela../desktop/git2020/applieddbdoodle/db_project/mongo.json*

- `$cd /usr/local/bin/` to get to folder where the `mongoimport` is.
- Then enter the following command to import the database: mongoimport `--db=proj20DB --collection=docs --file=/Users/ange../desktop/git2020/applieddbdoodle/db_project/mongo.json` --db=proj20DB --collection=docs --file=/Users/angelacorkery/desktop/git2020/applieddbdoodle/db_project/mongo.json

---

```
use proj20DB
```

> switched to db proj20DB

```
db
```

> proj20DB

```
show collections
```

> docs

```
db.docs.find({})
```

```
{ "_id" : "ARTS", "name" : "B.A.", "level" : 8 }
{ "_id" : "SW", "name" : "H. Dip. in SW Devel", "level" : 8 }
{ "_id" : "DATA", "name" : "H. Dip. in Data Analytics", "level" : 8 }
{ "_id" : "ENG", "name" : "B.Eng.", "level" : 7 }
{ "_id" : "G0033333", "details" : { "name" : "Tom Kenna", "address" :
"Dublin", "age" : 20 } }
{ "_id" : "G0012345", "details" : { "name" : "John Smith", "address" :
"Tuam", "age" : 21 }, "qualifications" : [ "ARTS", "DATA" ] }
```

```
{ "_id" : "G0022222", "details" : { "name" : "Mary Murphy", "address" :
"Castlebar", "age" : 21 }, "qualifications" : [ "ARTS" ] }
{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }
{ "_id" : "G0077777", "details" : { "name" : "Mick O'Hara", "address" :
"Ballinasloe", "age" : 41 }, "qualifications" : [ "ENG", "DATA", "SW" ] }
{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }
{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" :
"Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }
```

---

### 4.2.1 Average Age of Students

Give the MongoDB command to find the average age of students.

> { "Average" : 33 }

My answer uses the following:

```
> db.docs.aggregate([{$group:{_id:null,"Average":{$avg:"$details.age"}}},{
$unset: ["_id"] }])
```

{ "Average" : 33 }

There are 7 students in this database. `db.docs.count()` returns 11 documents. There are 7 documents for each student and 4 documents with details of the courses. It looks each student document contains an embedded document with the student details and also a reference to another document for their qualifications.

`db.docs.getIndexes()`

> [ { "v" : 2, "key" : { "_id" : 1 }, "name" : "*id*", "ns" : "proj20DB.docs" } ]

**A document for a student looks like this**

{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" : "Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }

**A document for a course looks like this**

{ "_id" : "ARTS", "name" : "B.A.", "level" : 8 }

db.student.find({},{address:1}) db.docs.find({},{details:1})

db.student.find({address:{$exists:true}},{_id:0,"address.county":1}) db.docs.find({details:{$exists:true}})

```
db.docs.find({details:{$exists:true}}).count()
```

returns 7

```
db.docs.find({details:{$exists:true}})
```

```
{ "_id" : "G0033333", "details" : { "name" : "Tom Kenna", "address" :
"Dublin", "age" : 20 } }

{ "_id" : "G0012345", "details" : { "name" : "John Smith", "address" :
"Tuam", "age" : 21 }, "qualifications" : [ "ARTS", "DATA" ] }

{ "_id" : "G0022222", "details" : { "name" : "Mary Murphy", "address" :
"Castlebar", "age" : 21 }, "qualifications" : [ "ARTS" ] }

{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }

{ "_id" : "G0077777", "details" : { "name" : "Mick O'Hara", "address" :
"Ballinasloe", "age" : 41 }, "qualifications" : [ "ENG", "DATA", "SW" ] }

{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }

{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" :
"Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }
```

Now narrowing down the query to just return the age.

```
db.docs.find({details:{$exists:true}},{_id:0,"details.age":1})
db.docs.find({"details.age":{$exists:true}},{_id:0,"details.age":1})

{ "details" : { "age" : 20 } }
{ "details" : { "age" : 21 } }
{ "details" : { "age" : 21 } }
{ "details" : { "age" : 53 } }
{ "details" : { "age" : 41 } }
{ "details" : { "age" : 35 } }
{ "details" : { "age" : 40 } }
```

returning only those over 40 years.

```
db.docs.find({"details.age":{$gt:30}},{_id:0,"details.age":1})
```

```
{ "details" : { "age" : 53 } } { "details" : { "age" : 41 } } { "details" : { "age" : 35 } } { "details" : { "age" : 40 } }
```

Example.

`bhp` is a sub-document or embedded document of the `engine` document so have to use "engine.bhp" using dot notation and quotes.

The following query returns the average bhp for each model:

```
db.docs.aggregate([{$group:{_id:"$model","Avg BHP":
{$avg:"$engine.bhp"}}}])
```

> { "_id" : "Fiesta", "Avg BHP" : 88 } { "_id" : "Mondeo", "Avg BHP" : 120 } { "_id" : "Corolla", "Avg BHP" : 118 } { "_id" : "Prius", "Avg BHP" : 90 } { "_id" : "Focus", "Avg BHP" : 118 }

(Add these to my notes on Github)

## Aggregation Pipeline Quick Reference

See Aggregation Pipeline Quick Reference

The aggregate method takes one parameter which is an array. When using aggregate you must use a $ in front of the field.

In the `db.collection.aggregate` method, pipeline stages appear in an array. Documents pass through the stages in sequence. All except the $out, $merge, and $geoNear stages can appear multiple times in a pipeline.

- `$group` is an aggregate pipeline stage. `$group` groups input documents by a specified identifier expression and applies the accumulator expression(s), if specified, to each group. Consumes all input documents and outputs one document per each distinct group. The output documents only contain the identifier field and, if specified, accumulated fields.
- `$lookup` is an aggregate pipeline stage. `$lookup` performs a left outer join to another collection in the same database to filter in documents from the “joined” collection for processing.
- `$project` is an aggregate pipeline stage. `$project` reshapes each document in the stream, such as by adding new fields or removing existing fields. For each input document, outputs one document.

nearly there...

```
db.docs.aggregate([{$group:{_id:null,"Average Age":
{$avg:"$details.age"}}}])
```

> { "_id" : null, "Average Age" : 33

```
db.docs.aggregate([{$group:{_id:"age","Average Age":
{$avg:"$details.age"}}}])
db.march.aggregate([{$group : {_id:"$age", "Max GPA per Age":
{$max:"$gpa"}}},{$sort:{_id:1}}])
db.docs.aggregate([{$group:{_id:null,"Average Age":
{$avg:"$details.age"}}},{ $unset: ["_id"] }])
```

This works!

> db.docs.aggregate([{$group:{_id:null,"Average":{$avg:"$details.age"}}},{ $unset: ["_id"] }]) { "Average" : 33 }

Just playing around...

```
db.docs.find({"details.age":{$gt:30}},{_id:0,"details.age":1})
db.docs.find({details:{$exists:true}},{_id:0, "details.age":1})
```

db.docs.aggregate([{$group:{_id:null,"Average":{$avg:"$details.age"}}},{ $project: {"_id":0} }])

db.student.find({_id:"G00101224"},{"modules.module":1}

db.docs.aggregate([{$lookup:{from:"docs" ,localField:"author", foreignField:"_id",as:"Written by"}}])

---

### 4.2.2 Honours Level

Give the MongoDB command to show the name of each course and Honours which has the value true if the course level is 8 or higher, otherwise false. The output should be sorted by name.

> { "name" : "B.A.", "Honours" : true } { "name" : "B.Eng.", "Honours" : false } { "name" : "H. Dip. in Data Analytics", "Honours" : true } { "name" : "H. Dip. in SW Devel", "Honours" : true }

**A document for a student looks like this**

{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" : "Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }

**A document for a course looks like this**

{ "_id" : "ARTS", "name" : "B.A.", "level" : 8 }

Here the "qualifications" field inside a students document refers to a referenced document.

Will be using the `$lookup` aggregation stage here. Actually don't need to as the question doesn't need any of the student details, only details about the course itself being level 8 or not.

First using a find just to see what is returned:

```
db.docs.find({level:{$exists:true}},{_id:false})

{ "name" : "B.A.", "level" : 8 }
{ "name" : "H. Dip. in SW Devel", "level" : 8 }
{ "name" : "H. Dip. in Data Analytics", "level" : 8 }
{ "name" : "B.Eng.", "level" : 7 }
```

### Using aggregation method

using `$match` and `$exists` pipeline stages to just return documents containing the course levels.

```
db.docs.aggregate({$match: {level:{$exists:true}}})

{ "_id" : "ARTS", "name" : "B.A.", "level" : 8 }
{ "_id" : "SW", "name" : "H. Dip. in SW Devel", "level" : 8 }
{ "_id" : "DATA", "name" : "H. Dip. in Data Analytics", "level" : 8 }
{ "_id" : "ENG", "name" : "B.Eng.", "level" : 7 }
```

also use the `$project` pipeline stage to limit what it returned in the query. The `_id` should be set to 0 so that it is not returned.

Need to use an aggregation operator on the level which is returned from the pipeline stage. Look at using comparison expression operators to look at the level, check if it level 8 or not which will return `true` if the level is 8, otherwise returns `false`.

```
db.docs.aggregate([{$match: {level:{$exists:true}}}, {$project:{_id:0}}])

{ "name" : "B.A.", "level" : 8 }
{ "name" : "H. Dip. in SW Devel", "level" : 8 }
{ "name" : "H. Dip. in Data Analytics", "level" : 8 }
{ "name" : "B.Eng.", "level" : 7 }
```

```
db.docs.aggregate([{$match: {level:{$exists:true}}}, {$project:{_id:0,
name:1, "Honours":{ $eq:["$level", 8]}}}])
{ "name" : "B.A.", "Honours" : true }
{ "name" : "H. Dip. in SW Devel", "Honours" : true }
{ "name" : "H. Dip. in Data Analytics", "Honours" : true }
{ "name" : "B.Eng.", "Honours" : false }
```

Getting there! need to sort the results by the name of the course.

```
 db.docs.aggregate([{$match: {level:{$exists:true}}}, {$project:{_id:0,
name:1, "Honours":{ $eq:["$level", 8]}}},{$sort:{name:-1}}])
{ "name" : "B.A.", "Honours" : true }
{ "name" : "B.Eng.", "Honours" : false }
{ "name" : "H. Dip. in Data Analytics", "Honours" : true }
{ "name" : "H. Dip. in SW Devel", "Honours" : true }
```

---

### 4.2.3 Qualified Students

Give the MongoDB command to show the number of Qualified Students i.e. those documents with a qualifications field.

{ "Qualified Students" : 6 }

```
 db.docs.find({})
{ "_id" : "ARTS", "name" : "B.A.", "level" : 8 }
{ "_id" : "SW", "name" : "H. Dip. in SW Devel", "level" : 8 }
{ "_id" : "DATA", "name" : "H. Dip. in Data Analytics", "level" : 8 }
{ "_id" : "ENG", "name" : "B.Eng.", "level" : 7 }
{ "_id" : "G0033333", "details" : { "name" : "Tom Kenna", "address" :
"Dublin", "age" : 20 } }
{ "_id" : "G0012345", "details" : { "name" : "John Smith", "address" :
"Tuam", "age" : 21 }, "qualifications" : [ "ARTS", "DATA" ] }
{ "_id" : "G0022222", "details" : { "name" : "Mary Murphy", "address" :
"Castlebar", "age" : 21 }, "qualifications" : [ "ARTS" ] }
{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }
{ "_id" : "G0077777", "details" : { "name" : "Mick O'Hara", "address" :
"Ballinasloe", "age" : 41 }, "qualifications" : [ "ENG", "DATA", "SW" ] }
{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }
{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" :
"Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }
```

The `qualifications` field is a referenced document. We don't actually need to refer to the referenced document, just check if this field exists.

First using a find to see what is returned for all documents containing a qualifications field.

```
db.docs.find({qualifications:{$exists:true}})


{ "_id" : "G0012345", "details" : { "name" : "John Smith", "address" :
"Tuam", "age" : 21 }, "qualifications" : [ "ARTS", "DATA" ] }
{ "_id" : "G0022222", "details" : { "name" : "Mary Murphy", "address" :
"Castlebar", "age" : 21 }, "qualifications" : [ "ARTS" ] }
{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }
{ "_id" : "G0077777", "details" : { "name" : "Mick O'Hara", "address" :
"Ballinasloe", "age" : 41 }, "qualifications" : [ "ENG", "DATA", "SW" ] }
{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }
{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" :
"Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }
```

This returns 6 documents.

```
db.docs.find({qualifications:{$exists:true}}).count()

6
```

Here I think a sum or count aggregation operator will be used. First need to work on the aggregation pipeline stages to limit what going into the operation.

```
db.docs.aggregate([{$match: {qualifications:{$exists:true}}}])
{ "_id" : "G0012345", "details" : { "name" : "John Smith", "address" :
"Tuam", "age" : 21 }, "qualifications" : [ "ARTS", "DATA" ] }
{ "_id" : "G0022222", "details" : { "name" : "Mary Murphy", "address" :
"Castlebar", "age" : 21 }, "qualifications" : [ "ARTS" ] }
{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }
{ "_id" : "G0077777", "details" : { "name" : "Mick O'Hara", "address" :
"Ballinasloe", "age" : 41 }, "qualifications" : [ "ENG", "DATA", "SW" ] }
{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }
{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" :
"Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }
```

We don't want the `details` attributes returned, or the _id so using the `$project` pipeline stage to exclude those fields.

```
db.docs.aggregate([{$match: {qualifications:{$exists:true}}}, {$project:
{_id:0, details:0}}])
{ "qualifications" : [ "ARTS", "DATA" ] }
{ "qualifications" : [ "ARTS" ] }
{ "qualifications" : [ "ENG, SW" ] }
{ "qualifications" : [ "ENG", "DATA", "SW" ] }
{ "qualifications" : [ "ENG", "SW" ] }
{ "qualifications" : [ "ARTS" ] }
```

db.docs.aggregate([{$match: {level:{$exists:true}}}, {$project:{_id:0, name:1, "Honours":{ $eq:["$level", 8]}}}, {$sort:{name:1}}])

First using the $match aggregation stage to filter the documents included to only those with the "qualifications" attribute. Then using the `$group` aggregation stage to group the results. Using `_id:null` so that the $group stage calculates accumulated values for all the input documents as a whole. The operation performed on the group is the sum of the documents that passed the match aggregation stage. Then using the `$project` stage to not return the `_id` attribute.

```
db.docs.aggregate( [
    {$match: {qualifications:{$exists:true}}},
    {$group: { _id: null, "Qualified Students": {$sum: 1} } },
    {$project: {_id:0}}  ])
```

Another simpler way is to use the `$count` aggregation stage which according to the documentation is equivalent to the $group and $project sequence I used above!

```
db.collection.aggregate( [
   { $group: { _id: null, myCount: { $sum: 1 } } },
   { $project: { _id: 0 } }
] )
```

```
 db.docs.aggregate( [ {$match: {qualifications:{$exists:true}}},  {
$group: { _id: null, "Qualified Students": {$sum: 1} } }, {$project:
{_id:0}}  ])


{ "Qualified Students" : 6 }
```

The $count pipeline stage passes a document to the next stage that contains a count of the number of documents input to the stage.

```
 db.docs.aggregate([{$match: { qualifications: {$exists: true}}}, {$count:
"Qualified Students"}] )


{ "Qualified Students" : 6 }
```

---

### 4.2.4 Students and their Qualifications

Give the MongoDB command to show the name of each Student and his/her qualifications. The output should be in alphabetical name order. If the student has no qualifications the word “None” should appear:

{ "details" : { "name" : "Alan Higgins" }, "qualifications" : [ "ENG, SW" ] } { "details" : { "name" : "Bernie Lynch" }, "qualifications" : [ "ARTS" ] } { "details" : { "name" : "Brian Collins" }, "qualifications" : [ "ENG", "SW" ] } { "details" : { "name" : "John Smith" }, "qualifications" : [ "ARTS", "DATA" ] } { "details" : { "name" : "Mary Murphy" }, "qualifications" : [ "ARTS" ] } { "details" : { "name" : "Mick O'Hara" }, "qualifications" : [ "ENG", "DATA", "SW" ] } { "details" : { "name" : "Tom Kenna" }, "qualifications" : "None"

To refresh:

```
 db.docs.find({})
{ "_id" : "ARTS", "name" : "B.A.", "level" : 8 }
{ "_id" : "SW", "name" : "H. Dip. in SW Devel", "level" : 8 }
{ "_id" : "DATA", "name" : "H. Dip. in Data Analytics", "level" : 8 }
{ "_id" : "ENG", "name" : "B.Eng.", "level" : 7 }
{ "_id" : "G0033333", "details" : { "name" : "Tom Kenna", "address" :
"Dublin", "age" : 20 } }
{ "_id" : "G0012345", "details" : { "name" : "John Smith", "address" :
"Tuam", "age" : 21 }, "qualifications" : [ "ARTS", "DATA" ] }
{ "_id" : "G0022222", "details" : { "name" : "Mary Murphy", "address" :
"Castlebar", "age" : 21 }, "qualifications" : [ "ARTS" ] }
{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }
{ "_id" : "G0077777", "details" : { "name" : "Mick O'Hara", "address" :
"Ballinasloe", "age" : 41 }, "qualifications" : [ "ENG", "DATA", "SW" ] }
{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }
{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" :
"Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }
```

The name of each student is inside of a sub-document or embedded document in the "details" attribute. The qualifications of each student use document references to the qualifications documents.

```
db.docs.aggregate([{$match: { details: {$exists: true}}}] )
```

To get information from documents with document references, need to use the aggregation method, using the pipeline stage `$lookup` which has 4 fields. `$lookup` performs a left outer join to another collection in the same database to filter in documents from the “joined” collection for processing.

The $pipeline stage `$lookup` can be thought of as similar to a `join` in MySQL. With the `$lookup` pipeline stage, `from` is the collection in which to perform the lookup, `localField` is the value(s) being searched for, `foreignField` is the field to search for the `localField` value(s) in and `as` is any string provided as the name of the output.

The $lookup stage has the following syntax:

```
{
   $lookup:
     {
       from: <collection to join>,
       localField: <field from the input documents>,
       foreignField: <field from the documents of the "from" collection>,
       as: <output array field>
     }
}
```

- `from` specifies the collection in the same database to perform the join with. **docs**
- `localField` is the value to search for. **qualifications**
- `foreignField` is the field to search for the value specified by localField. **_id**
- `as` is the name of the output. (i.e the name of fields containing the fields of the lookup). **name**

```
db.docs.find({details:{$exists:true}},{_id:0,"details.name":1,
"qualifications":1}).sort({"details.name":1})
```

{ "details" : { "name" : "Alan Higgins" }, "qualifications" : [ "ENG, SW" ] } { "details" : { "name" : "Bernie Lynch" }, "qualifications" : [ "ARTS" ] } { "details" : { "name" : "Brian Collins" }, "qualifications" : [ "ENG", "SW" ] } { "details" : { "name" : "John Smith" }, "qualifications" : [ "ARTS", "DATA" ] } { "details" : { "name" : "Mary Murphy" }, "qualifications" : [ "ARTS" ] } { "details" : { "name" : "Mick O'Hara" }, "qualifications" : [ "ENG", "DATA", "SW" ] } { "details" : { "name" : "Tom Kenna" } }

Getting close! need to return "None" for Tom who does not have a qualification. I think I need to do an aggregation for this?

```
db.docs.aggregate( [
    {$match: {details:{$exists:true}}},
    {$lookup:{from:"docs" ,localField:"qualifications",
foreignField:"_id",as:"qualifications"}},
    {$project: {_id:0, "details.address":0, "details.age":0}}] )
```

Starting with a find to look at the documents. Using the `find(query, projection)` method here with the query being the details field. checking if the details field exists in a document. The projection part is specifying what attributes to return. The results are then sorted by the persons name which is in the details embedded document so "details.name"

```
 db.docs.aggregate( [
... {$match: {details:{$exists:true}}},
... {$lookup:{from:"docs" ,localField:"qualifications",
foreignField:"_id",as:"qualifications"}},
... {$project: {_id:0, "details.address":0, "details.age":0}}] )


{ "details" : { "name" : "Tom Kenna" }, "qualifications" : [ ] }

{ "details" : { "name" : "John Smith" }, "qualifications" : [ { "_id" :
"ARTS", "name" : "B.A.", "level" : 8 }, { "_id" : "DATA", "name" : "H.
Dip. in Data Analytics", "level" : 8 } ] }

{ "details" : { "name" : "Mary Murphy" }, "qualifications" : [ { "_id" :
"ARTS", "name" : "B.A.", "level" : 8 } ] }

{ "details" : { "name" : "Alan Higgins" }, "qualifications" : [ ] }

{ "details" : { "name" : "Mick O'Hara" }, "qualifications" : [ { "_id" :
```

```
"DATA", "name" : "H. Dip. in Data Analytics", "level" : 8 }, { "_id" :
"ENG", "name" : "B.Eng.", "level" : 7 }, { "_id" : "SW", "name" : "H. Dip.
in SW Devel", "level" : 8 } ] }

{ "details" : { "name" : "Brian Collins" }, "qualifications" : [ { "_id" :
"ENG", "name" : "B.Eng.", "level" : 7 }, { "_id" : "SW", "name" : "H. Dip.
in SW Devel", "level" : 8 } ] }

{ "details" : { "name" : "Bernie Lynch" }, "qualifications" : [ { "_id" :
"ARTS", "name" : "B.A.", "level" : 8 } ] }
>
```

This is what the documents for Alan Higgins and Brian Collins looks like.

```
{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }
{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }
```

Yet when I do an aggregation, Alan's qualification returns an empty array unlike Brian's.??

```
db.docs.aggregate( [
    {$match: {details:{$exists:true}}},
    {$lookup:{from:"docs", localField:"qualifications",
foreignField:"_id",as:"qualifications"}}] )
```

```
db.docs.aggregate([{$match: {details:{$exists:true}}}, {$project:{_id:0,
"details.address":0, "details.age":0}},{$sort:{"details.name":1}}])

 db.docs.aggregate(
 [
    {$match: {details:{$exists:true}}},
    {$project:{_id:0, "details.address":0, "details.age":0,
"qualifications":{ $ifNull: [ "qualifications", "None" ]}},
    {$sort:{"details.name":1}}])

db.docs.aggregate([
    {$match: {details:{$exists:true}}},
    {$lookup:{from:"docs", localField:"qualifications",
foreignField:"_id",as:"myqualifications"}},
    {$sort:{"details.name":1}}
    ])
```

```json
 db.docs.find({})
{ "_id" : "ARTS", "name" : "B.A.", "level" : 8 }
{ "_id" : "SW", "name" : "H. Dip. in SW Devel", "level" : 8 }
{ "_id" : "DATA", "name" : "H. Dip. in Data Analytics", "level" : 8 }
{ "_id" : "ENG", "name" : "B.Eng.", "level" : 7 }
{ "_id" : "G0033333", "details" : { "name" : "Tom Kenna", "address" :
"Dublin", "age" : 20 } }
{ "_id" : "G0012345", "details" : { "name" : "John Smith", "address" :
"Tuam", "age" : 21 }, "qualifications" : [ "ARTS", "DATA" ] }
{ "_id" : "G0022222", "details" : { "name" : "Mary Murphy", "address" :
"Castlebar", "age" : 21 }, "qualifications" : [ "ARTS" ] }
{ "_id" : "G0066666", "details" : { "name" : "Alan Higgins", "address" :
"Galway", "age" : 53 }, "qualifications" : [ "ENG, SW" ] }
{ "_id" : "G0077777", "details" : { "name" : "Mick O'Hara", "address" :
"Ballinasloe", "age" : 41 }, "qualifications" : [ "ENG", "DATA", "SW" ] }
{ "_id" : "G0044444", "details" : { "name" : "Brian Collins", "address" :
"Dublin", "age" : 35 }, "qualifications" : [ "ENG", "SW" ] }
{ "_id" : "G0055555", "details" : { "name" : "Bernie Lynch", "address" :
"Galway", "age" : 40 }, "qualifications" : [ "ARTS" ] }
```

```
db.docs.aggregate([{$match: {details:{$exists:true}}}, {$sort:
{"details.name": 1}},
{$project:{"details.name":1,"details.address":0, "details.age":0}}
    ])

db.docs.aggregate([{$match: {details:{$exists:true}}}, {$sort:
{"details.name": 1}}, {$project:{"details.address":0,
_id:0,"details.age":0, qualifications:0}}])
```

I want to return the qualifications field with none if it doesn't exist.

You cannot include and exlude in the one $project aggregation stage (apart from excluding _id). I want to exclude details.age and details.address but include the qualifications attribute.

```
db.docs.aggregate([{$match: {details:{$exists:true}}},
{$project:{"details.address":0,"details.age":0, _id:0}},{$project:
{qualifications:1}},{$sort:{"details.name": 1}}])

{ "details" : { "name" : "Alan Higgins" }, "qualifications" : [ "ENG, SW"
] }
{ "details" : { "name" : "Bernie Lynch" }, "qualifications" : [ "ARTS" ] }
{ "details" : { "name" : "Brian Collins" }, "qualifications" : [ "ENG",
"SW" ] }
{ "details" : { "name" : "John Smith" }, "qualifications" : [ "ARTS",
"DATA" ] }
{ "details" : { "name" : "Mary Murphy" }, "qualifications" : [ "ARTS" ] }
```

```
{ "details" : { "name" : "Mick O'Hara" }, "qualifications" : [ "ENG",
"DATA", "SW" ] }
{ "details" : { "name" : "Tom Kenna" } }
```

Can use $ifNull aggregation pipeline operator to return a value for a field if it doesn't exist. Do this inside a $project aggregation pipeline stage.

Initially I was trying to exclude the address and age and include qualifications at the same time using a single $project pipeline stage.

```
 db.docs.aggregate([{$match: {details:{$exists:true}}},{$project:
{"details.name":1,qualifications:1, _id:0}},{$sort:{"details.name": 1}}])
{ "details" : { "name" : "Alan Higgins" }, "qualifications" : [ "ENG, SW"
] }
{ "details" : { "name" : "Bernie Lynch" }, "qualifications" : [ "ARTS" ] }
{ "details" : { "name" : "Brian Collins" }, "qualifications" : [ "ENG",
"SW" ] }
{ "details" : { "name" : "John Smith" }, "qualifications" : [ "ARTS",
"DATA" ] }
{ "details" : { "name" : "Mary Murphy" }, "qualifications" : [ "ARTS" ] }
{ "details" : { "name" : "Mick O'Hara" }, "qualifications" : [ "ENG",
"DATA", "SW" ] }
{ "details" : { "name" : "Tom Kenna" } }
```

```
 db.docs.aggregate([{$match: {details:{$exists:true}}},{$project:
{"details.name":1,_id:0,qualifications:{ $ifNull:
["$qualifications","None"]}}},{$sort:{"details.name": 1}}])


{ "details" : { "name" : "Alan Higgins" }, "qualifications" : [ "ENG, SW"
] }
{ "details" : { "name" : "Bernie Lynch" }, "qualifications" : [ "ARTS" ] }
{ "details" : { "name" : "Brian Collins" }, "qualifications" : [ "ENG",
"SW" ] }
{ "details" : { "name" : "John Smith" }, "qualifications" : [ "ARTS",
"DATA" ] }
{ "details" : { "name" : "Mary Murphy" }, "qualifications" : [ "ARTS" ] }
{ "details" : { "name" : "Mick O'Hara" }, "qualifications" : [ "ENG",
"DATA", "SW" ] }
{ "details" : { "name" : "Tom Kenna" }, "qualifications" : "None" }
```