MongoDB can compute aggregate statistics such as sum, average, minimum, maximum, using the method:

```
collection.aggregate(pipeline)
```

A pipeline is a list of ordered operations.
Order matters: each stage receives the result of the previous one.

Main Aggregation Operators

Operator	Meaning
$limit	Limit the number of documents
$sort	Sort documents
$match	Filter documents
$unwind	Expand an array into multiple documents
$addFields	Add or modify fields
$project	Reshape documents (keep/remove/rename fields)
$group	Group documents and compute aggregates
$sortByCount	Group + count + sort
$lookup	Join with another collection

Setup in Python

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017/")
db = client["test"]
restaurants = db["restaurants"]
```

1. **$limit**

Limit display to the first 10 restaurants:

```
pipeline = [
    {"$limit": 10}
]

for doc in restaurants.aggregate(pipeline):
    print(doc)
```

2. **$sort**

Sort by restaurant name:

```
pipeline = [
    {"$limit": 10},
    {"$sort": {"name": 1}}   # 1 = ascending, -1 = descending
]

for doc in restaurants.aggregate(pipeline):
    print(doc)
```

3. $match

Filter restaurants located in Brooklyn:

```
pipeline = [
    {"$limit": 10},
    {"$sort": {"name": 1}},
    {"$match": {"borough": "Brooklyn"}}
]

for doc in restaurants.aggregate(pipeline):
    print(doc)
```

Filtering **before** limiting (better logic):

```
pipeline = [
    {"$match": {"borough": "Brooklyn"}},
    {"$limit": 10},
    {"$sort": {"name": 1}}
]
```

4. $unwind

The purpose of the $unwind operation is to *explode* an array inside a document.

- A document containing an array of n elements becomes n documents.
- Each resulting document contains one element of the original array instead of the whole array.
- You must specify the name of the array field (with a $ prefix).

4.1. Unwind the grades array for the first 10 restaurants

```
pipeline = [
    {"$limit": 10},
    {"$unwind": "$grades"}
]

for doc in restaurants.aggregate(pipeline):
    print(doc)
```

4.2. Same as above, but keep only restaurants whose grade is “B”

```
pipeline = [
    {"$limit": 10},
    {"$unwind": "$grades"},
    {"$match": {"grades.grade": "B"}}
]
```

```
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

4.3. If we reverse $unwind and $match, the result changes

```
pipeline = [
    {"$limit": 10},
    {"$match": {"grades.grade": "B"}},
    {"$unwind": "$grades"}
]

for doc in restaurants.aggregate(pipeline):
    print(doc)
```

- in this order, $match tries to find restaurants whose grades array contains "B" anywhere (without splitting).
- After that, $unwind splits the grades array.
- So, restaurants with a “B” will appear, but you will see all grades, not only “B”.

5. $addFields and $project

$addFields
→ Adds new fields to the documents.

`$project`
→ Selects which fields to keep, remove, rename, or compute.

Useful projection operators:

- `$arrayElemAt → element of an array`
- `$first, $last → first or last element of an array`
- `$size → array length`
- `$substr → substring`
- `$cond → conditional (if-then-else)`

5.1. Add a field: number of evaluations (nb_grades)

```
pipeline = [
    {"$limit": 10},
    {"$addFields": {"nb_grades": {"$size": "$grades"}}}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.2. Keep only name and borough of first 10 restaurants

```
pipeline = [
    {"$limit": 10},
```

```
    {"$project": {"name": 1, "borough": 1}}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.3. Remove address and grades

```
pipeline = [
    {"$limit": 10},
    {"$project": {"address": 0, "grades": 0}}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.4. Add a new field street containing address.street

```
pipeline = [
    {"$limit": 10},
    {"$project": {"name": 1, "borough": 1, "street": "$address.street"}}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.5. Add number of visits (nb_grades = size of grades)

```
pipeline = [
    {"$limit": 10},
    {"$project": {"name": 1, "borough": 1, "nb_grades": {"$size":
"$grades"}}}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.6. Sort by number of visits and show first 10

```
pipeline = [
    {"$project": {"name": 1, "borough": 1, "nb_grades": {"$size":
"$grades"}}},
    {"$sort": {"nb_grades": 1}},
    {"$limit": 10}
]
```

```
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.7. Keep only the first element of the grades array

```
pipeline = [
    {"$limit": 10},
    {"$project": {"name": 1, "borough": 1, "grade": {"$arrayElemAt":
["$grades", 0]}}}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.8. Uppercase the restaurant name

```
pipeline = [
    {"$limit": 10},
    {"$project": {"nom": {"$toUpper": "$name"}, "borough": 1}}
]


for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.9. Extract the first 3 letters of borough

```
pipeline = [
    {"$limit": 10},
    {"$project": {
        "nom": {"$toUpper": "$name"},
        "quartier": {"$substr": ["$borough", 0, 3]}
    }}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

5.10. Replace borough abbreviation with "BRX" for Bronx using $cond

```
pipeline = [
    {"$limit": 10},
    {"$addFields": {
        "quartier": {"$toUpper": {"$substr": ["$borough", 0, 3]}}
    }},
```

```
    {"$project": {
        "nom": {"$toUpper": "$name"},
        "quartier": {
            "$cond": {
                "if": {"$eq": ["$borough", "Bronx"]},
                "then": "BRX",
                "else": "$quartier"
            }
        },
        "borough": 1
    }}
]
for doc in restaurants.aggregate(pipeline):
    print(doc)
```

6. **$group**

The $group operator is used to perform aggregations — just like in SQL (COUNT, SUM, AVG, etc.).

Meaning of _id in $group

_id defines the grouping key:

- String constant → no grouping (all documents in one group)
  Example: _id: "Total"
- Field name → group by that field
  Example: _id: "$borough"
- Object with several fields → multiple grouping criteria
  Example:

$sum

- Add values together
- If you sum 1, you get a count of documents
- If you sum a field, you get the sum of its values

$avg

Compute the average of a numeric field.

$min, $max

Find the minimum or maximum value in the group.

$addToSet

Collect distinct values into an array
(no duplicates).

$push

Collect all values into an array
(duplicates included).

6.1. Total number of restaurants

```
pipeline = [ {"$group": {"_id": "Total", "NbRestos": {"$sum": 1}}}]
result = list(restaurants.aggregate(pipeline))
print(result)
```

6.2. Number of restaurants per borough

```
pipeline = [{"$group": {"_id": "$borough", "NbRestos": {"$sum": 1}}}]
result = list(restaurants.aggregate(pipeline))
print(result)
```

6.3. Average score of restaurants in Queens

```
pipeline = [
    {"$match": {"borough": "Queens"}},
    {"$unwind": "$grades"},
    {"$group": {"_id": None, "score": {"$avg": "$grades.score"}}}]
result = list(restaurants.aggregate(pipeline))
print(result)
```

6.4. Average score by borough, sorted descending

```
pipeline = [
    {"$unwind": "$grades"},
    {"$group": {"_id": "$borough", "score": {"$avg": "$grades.score"}}},
    {"$sort": {"score": -1}}   ]
result = list(restaurants.aggregate(pipeline))
print(result)
```

6.5. Top 10 healthiest streets (borough + street)

```
pipeline = [
```

```
    {"$project": {
        "borough": 1,
        "street": "$address.street",
        "eval": {"$arrayElemAt": ["$grades", 0]}
    }},
    {"$match": {"eval": {"$exists": True}}},
    {"$match": {"eval.score": {"$gte": 0}}},
    {"$group": {
        "_id": {"quartier": "$borough", "rue": "$street"},
        "score": {"$avg": "$eval.score"}
    }},
    {"$sort": {"score": 1}},
    {"$limit": 10}
]
result = list(restaurants.aggregate(pipeline))
print(result)
```

- $project: keep borough, street, and take **first grade** ($arrayElemAt)
- Remove restaurants without grades
- Group by (borough, street)
- Sort by score (low score = healthy)

6.6. Difference between $addToSet and $push

```
pipeline = [
    {"$limit": 10},
    {"$unwind": "$grades"},
    {"$group": {
        "_id": "$name",
        "avec_addToSet": {"$addToSet": "$grades.grade"},
        "avec_push": {"$push": "$grades.grade"}
    }}
]
result = list(restaurants.aggregate(pipeline))
print(result)
```

- $addToSet: unique values only

- $push: all values, including duplicates

7. $sortByCount

The $sortByCount operator:

1. Groups documents by the specified field (use $ before the field name).
2. Counts the number of documents for each value of that field.
3. Sorts the results in descending order of the count.

This operator is very useful for generating TOP lists (most frequent values).

Example: Top boroughs in New York by restaurant count

```
pipeline = [  { "$sortByCount": "$borough" }]
result = restaurants.aggregate(pipeline)
for doc in result:
    print(doc)
```

Equivalent using $group + $sort:

```
pipeline = [
    {
        "$group": {
            "_id": "$borough",   # Group by borough
            "count": { "$sum": 1 }  # Count restaurants in each borough
        }
    },
    { "$sort": { "count": -1 } }  # Sort descending by count
]
result = restaurants.aggregate(pipeline)
for doc in result:
    print(doc)
```

To Do:

Connect to MongoDB

```
from pymongo import MongoClient
client = MongoClient("mongodb://localhost:27017/")
db = client["test"]
restaurants = db["restaurants"]
```

1. **What are the 10 largest restaurant chains** (restaurants with identical names)?
    - **Classic TOP 10** (two ways to do it).
2. Give the **Top 5** and **Flop 5 cuisine types** in terms of the number of restaurants.
    - Same task, but with **different sorting** for the two queries.
3. What are the **10 streets with the most restaurants**?
    - TOP 10 as well.
4. Which streets are located in **strictly more than 2 boroughs**?
    - Try to **add the names of the boroughs** for each street (see $addToSet).
5. List by **borough** the number of restaurants and the **average score**.
    - Be careful to properly **unwind the grades array**.
6. Give the **start and end dates** of evaluations.
    - Use **min** and **max**.
7. What are the **10 restaurants (name, borough, address, and score) with the lowest average score**?
    - Requires **unwinding, grouping by restaurant, sorting, and limiting**.
8. Which restaurants (name, borough, and address) have **only "A" grades**?
    - Restrict to those with "A", **unwind**, remove other grades besides "A", and display the info.
    - Other approaches: **unwind, addToSet**, and restrict to those where the resulting array = ["A"].
9. **Count the number of reviews per day of the week**
    - A small investigation is needed on how to extract the day of the week from a date.
10. **Give the 3 most common cuisine types per borough**
    - Easy to say, complicated to do

    A hint:

    - Double grouping will be needed
    - Sorting will be required
    - Grouping with $push
    - Use $slice to take a portion of an array