"UNDERSTANDING MONGODB AGGREGATION PIPELINE"

An aggregation pipeline in MongoDB is a way to process and analyze data by passing it through a series of stages. Each stage performs a specific operation on the data, like filtering, grouping, or sorting. The output of one stage becomes the input for the next stage, allowing you to transform the data step-by-step to get the desired results.

Here's a brief explanation of each stage in a MongoDB aggregation pipeline:

1. **$match:** Selects documents that match certain criteria.
2. **$group:**Groups documents by a field and performs calculations for each group.
3. **$sort**:Sorts documents by a specified field.
4. **$project:**Chooses which fields to include or create new fields.
5. **$skip:**Skips a specified number of documents.
6. **$limit:**Limits the number of documents to a specified number.
7. **$unwind**:Breaks apart an array field to create a separate document for each item.

These stages can be combined to process and analyze data step-by-step in MongoDB.
The following data set has been designated as a new collection named students6. This collection comprises records of 20 students, each with attributes such as ID, name, age, major, and scores. The structure of the data is outlined below:

```
[
  { "_id": 1, "name": "Alice", "age": 25, "major": "Computer Science", "scores": [85, 92, 78] },
  { "_id": 2, "name": "Bob", "age": 22, "major": "Mathematics", "scores": [90, 88, 95] },
  { "_id": 3, "name": "Charlie", "age": 28, "major": "English", "scores": [75, 82, 89] },
  { "_id": 4, "name": "David", "age": 20, "major": "Computer Science", "scores": [98, 95, 87] },
  { "_id": 5, "name": "Eve", "age": 23, "major": "Biology", "scores": [80, 77, 93] },
  { "_id": 6, "name": "Faythe", "age": 24, "major": "Chemistry", "scores": [88, 90, 92] },
  { "_id": 7, "name": "Grace", "age": 21, "major": "Physics", "scores": [91, 85, 87] },
  { "_id": 8, "name": "Heidi", "age": 26, "major": "Mathematics", "scores": [78, 82, 88] },
  { "_id": 9, "name": "Ivan", "age": 22, "major": "History", "scores": [84, 89, 91] },
  { "_id": 10, "name": "Judy", "age": 25, "major": "Biology", "scores": [79, 83, 85] },
  { "_id": 11, "name": "Kathy", "age": 23, "major": "Computer Science", "scores": [90, 94, 97] },
  { "_id": 12, "name": "Louis", "age": 27, "major": "Philosophy", "scores": [82, 80, 78] },
  { "_id": 13, "name": "Mallory", "age": 20, "major": "Physics", "scores": [86, 88, 92] },
  { "_id": 14, "name": "Niaj", "age": 24, "major": "English", "scores": [87, 90, 85] },
  { "_id": 15, "name": "Olivia", "age": 22, "major": "Mathematics", "scores": [93, 95, 90] },
  { "_id": 16, "name": "Peggy", "age": 21, "major": "Chemistry", "scores": [79, 85, 88] },
  { "_id": 17, "name": "Quentin", "age": 28, "major": "Computer Science", "scores": [85, 89, 92] },
  { "_id": 18, "name": "Rupert", "age": 25, "major": "Biology", "scores": [84, 88, 90] },
  { "_id": 19, "name": "Sybil", "age": 20, "major": "Physics", "scores": [90, 92, 95] },
  { "_id": 20, "name": "Trent", "age": 23, "major": "History", "scores": [81, 85, 87] }
]
```

Here is an expanded explanation for each stage along with the examples:

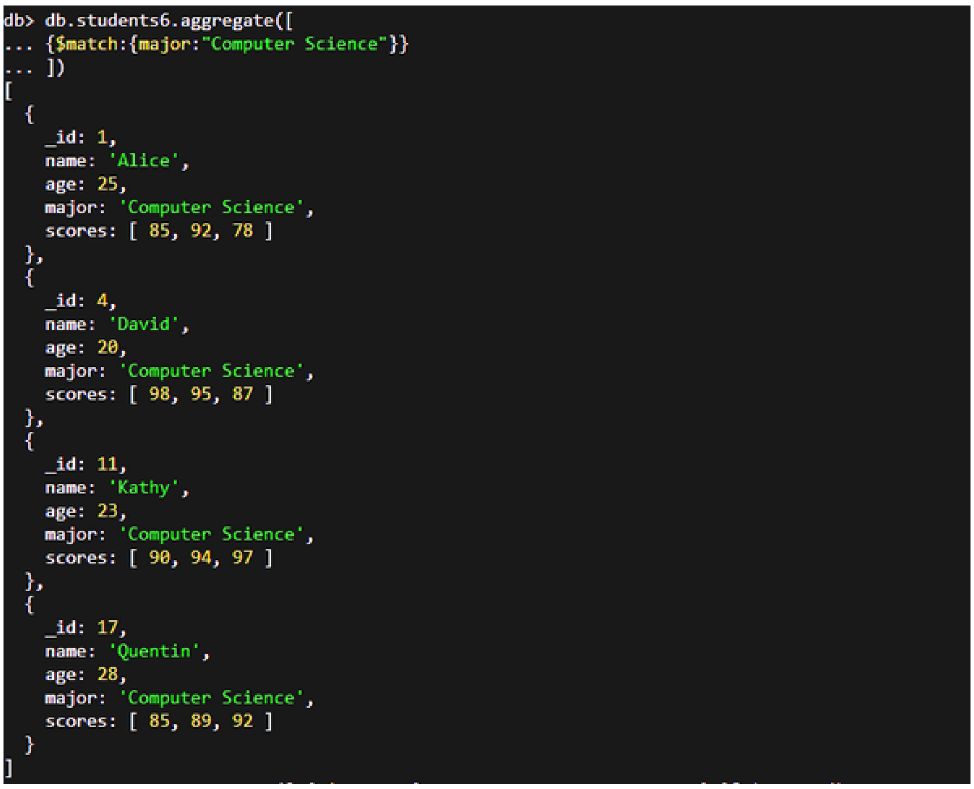
1. $MATCH:

The $match stage filters documents in the pipeline. Only documents that match the specified criteria are passed to the next stage. This stage is highly efficient when used early in the pipeline as it reduces the amount of data that needs to be processed by subsequent stages.

Example: Find students majoring in "Computer Science".

```
db.students6.aggregate([
  { $match: { major: "Computer Science" } }
])
```

This stage filters the documents to include only those where the major field is "Computer Science".



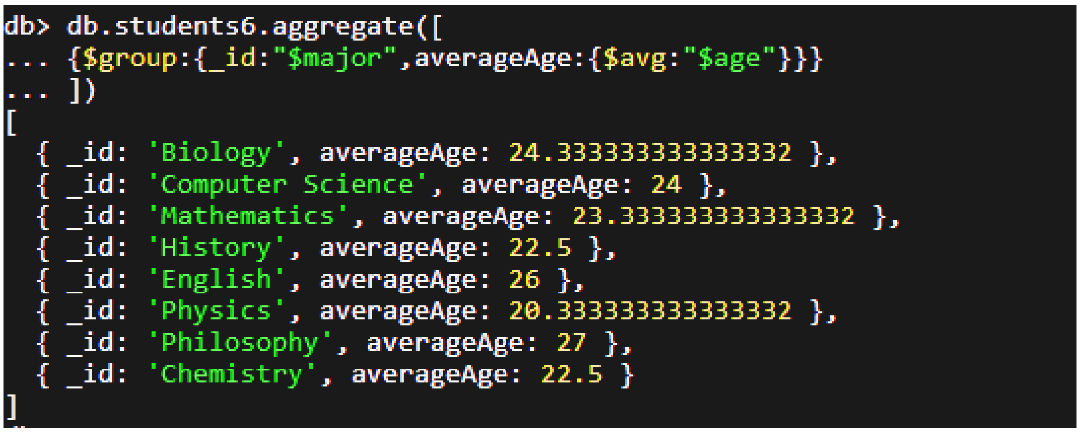
```
db> db.students6.aggregate([
... {$match:{major:"Computer Science"}}
... ])
[
  {
    _id: 1,
    name: 'Alice',
    age: 25,
    major: 'Computer Science',
    scores: [ 85, 92, 78 ]
  },
  {
    _id: 4,
    name: 'David',
    age: 20,
    major: 'Computer Science',
    scores: [ 98, 95, 87 ]
  },
  {
    _id: 11,
    name: 'Kathy',
    age: 23,
    major: 'Computer Science',
    scores: [ 90, 94, 97 ]
  },
  {
    _id: 17,
    name: 'Quentin',
    age: 28,
    major: 'Computer Science',
    scores: [ 85, 89, 92 ]
  }
]
```

2. $GROUP:

Groups documents by a specified key and performs aggregate operations.

Example: Group by major and calculate the average age of students in each major.

```
db.students6.aggregate([
  { $group: { _id: "$major", averageAge: { $avg: "$age" } } }
])
```



```
db> db.students6.aggregate([
... {$group:{_id:"$major",averageAge:{$avg:"$age"}}}
... ])
[
  { _id: 'Biology', averageAge: 24.333333333333332 },
  { _id: 'Computer Science', averageAge: 24 },
  { _id: 'Mathematics', averageAge: 23.333333333333332 },
  { _id: 'History', averageAge: 22.5 },
  { _id: 'English', averageAge: 26 },
  { _id: 'Physics', averageAge: 20.333333333333332 },
  { _id: 'Philosophy', averageAge: 27 },
  { _id: 'Chemistry', averageAge: 22.5 }
]
```

Explanation: This stage groups the documents by the major field and calculates the average age for each group.

3. $SORT

The $sort stage sorts all input documents and returns them in the specified order. The sort order is determined by the value of a specified field or fields. Sorting can be done in ascending or descending order and is useful for organizing output data.

4. $PROJECT:

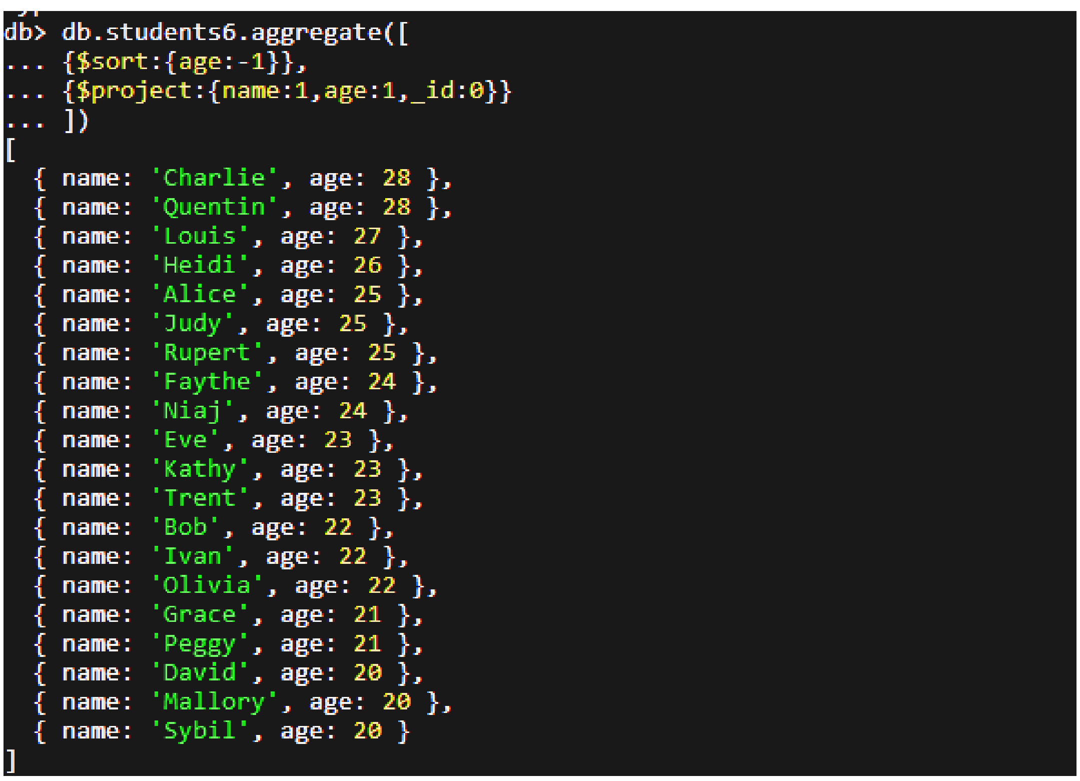
The $project stage reshapes each document in the pipeline. It can include, exclude, or add new fields to the documents. Computed fields can also be created using expressions. This is useful for transforming the structure of the documents and preparing the data for further stages or final output.

Example: Sort students by age in descending order.

```
db.students6.aggregate([
  { $sort: { age: -1 } },  // Step 1: Sort students by age in descending order
   { $project: { name: 1, age: 1, _id: 0 } }  // Step 2: Project only the name and age fields,
exclude _id
])
```

Explanation:

1. $sort: Sorts the students by age in descending order.
2. $project: Includes only the name and age fields and explicitly excludes the _id field by setting _id: 0.



```
db> db.students6.aggregate([
... {$sort:{age:-1}},
... {$project:{name:1,age:1,_id:0}}
... ])
[
  { name: 'Charlie', age: 28 },
  { name: 'Quentin', age: 28 },
  { name: 'Louis', age: 27 },
  { name: 'Heidi', age: 26 },
  { name: 'Alice', age: 25 },
  { name: 'Judy', age: 25 },
  { name: 'Rupert', age: 25 },
  { name: 'Faythe', age: 24 },
  { name: 'Niaj', age: 24 },
  { name: 'Eve', age: 23 },
  { name: 'Kathy', age: 23 },
  { name: 'Trent', age: 23 },
  { name: 'Bob', age: 22 },
  { name: 'Ivan', age: 22 },
  { name: 'Olivia', age: 22 },
  { name: 'Grace', age: 21 },
  { name: 'Peggy', age: 21 },
  { name: 'David', age: 20 },
  { name: 'Mallory', age: 20 },
  { name: 'Sybil', age: 20 }
]
```

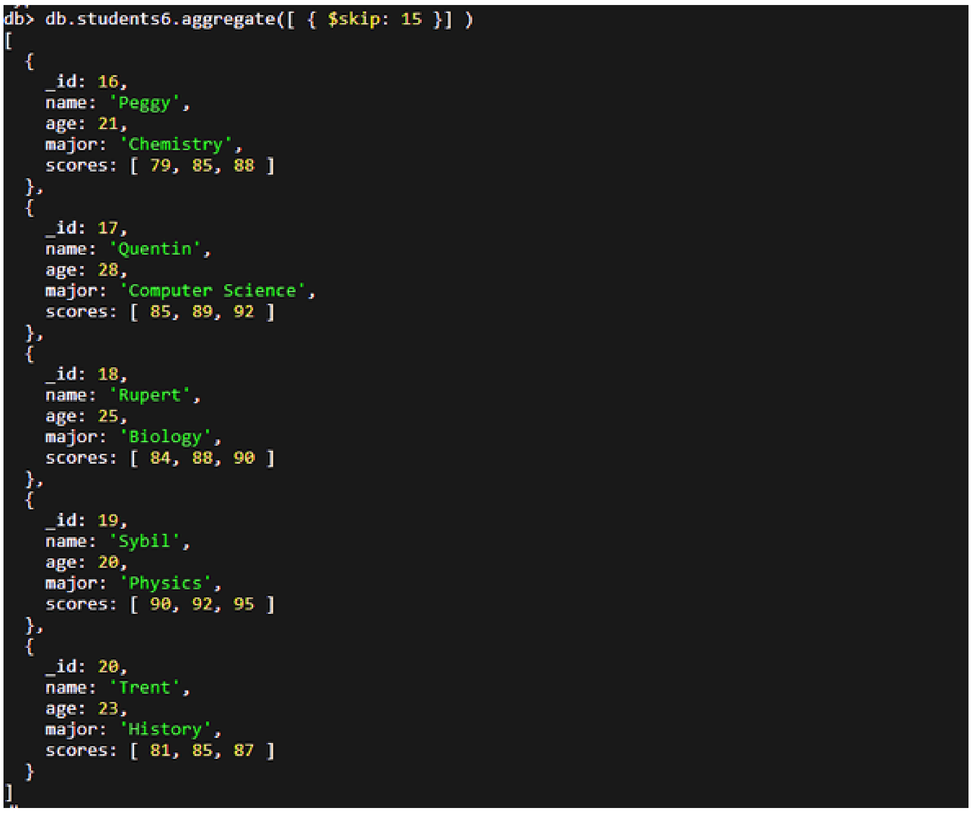
5. $SKIP

The $skip stage skips a specified number of documents in the pipeline. This is useful for pagination, where you might want to skip a number of documents to retrieve the next set of results.

Example: Skip the first 15 students.

```
db.students6.aggregate([
  { $skip: 15 }
])
```

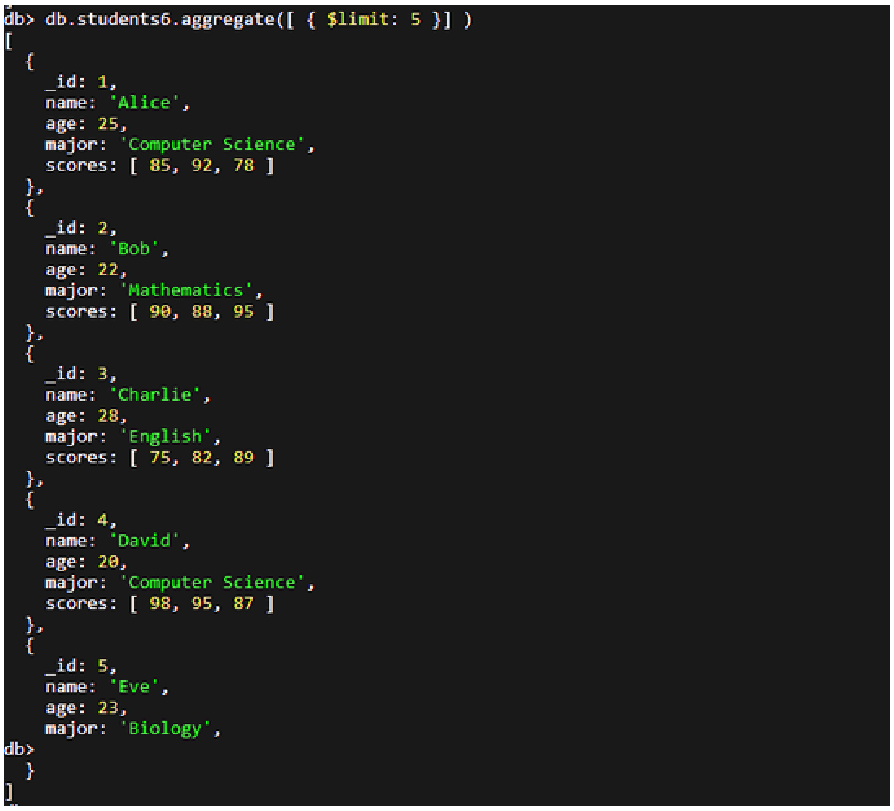
Explanation: This stage skips the first 15 documents in the pipeline.



```
db> db.students6.aggregate([ { $skip: 15 }] )
[
  {
    _id: 16,
    name: 'Peggy',
    age: 21,
    major: 'Chemistry',
    scores: [ 79, 85, 88 ]
  },
  {
    _id: 17,
    name: 'Quentin',
    age: 28,
    major: 'Computer Science',
    scores: [ 85, 89, 92 ]
  },
  {
    _id: 18,
    name: 'Rupert',
    age: 25,
    major: 'Biology',
    scores: [ 84, 88, 90 ]
  },
  {
    _id: 19,
    name: 'Sybil',
    age: 20,
    major: 'Physics',
    scores: [ 90, 92, 95 ]
  },
  {
    _id: 20,
    name: 'Trent',
    age: 23,
    major: 'History',
    scores: [ 81, 85, 87 ]
  }
]
```

6. $LIMIT:

The $limit stage limits the number of documents passed to the next stage. It is often used in conjunction with $skip to implement pagination. Limiting the number of documents can also help in reducing the processing load for subsequent stages.
Example: Limit the result to the first 10 students.

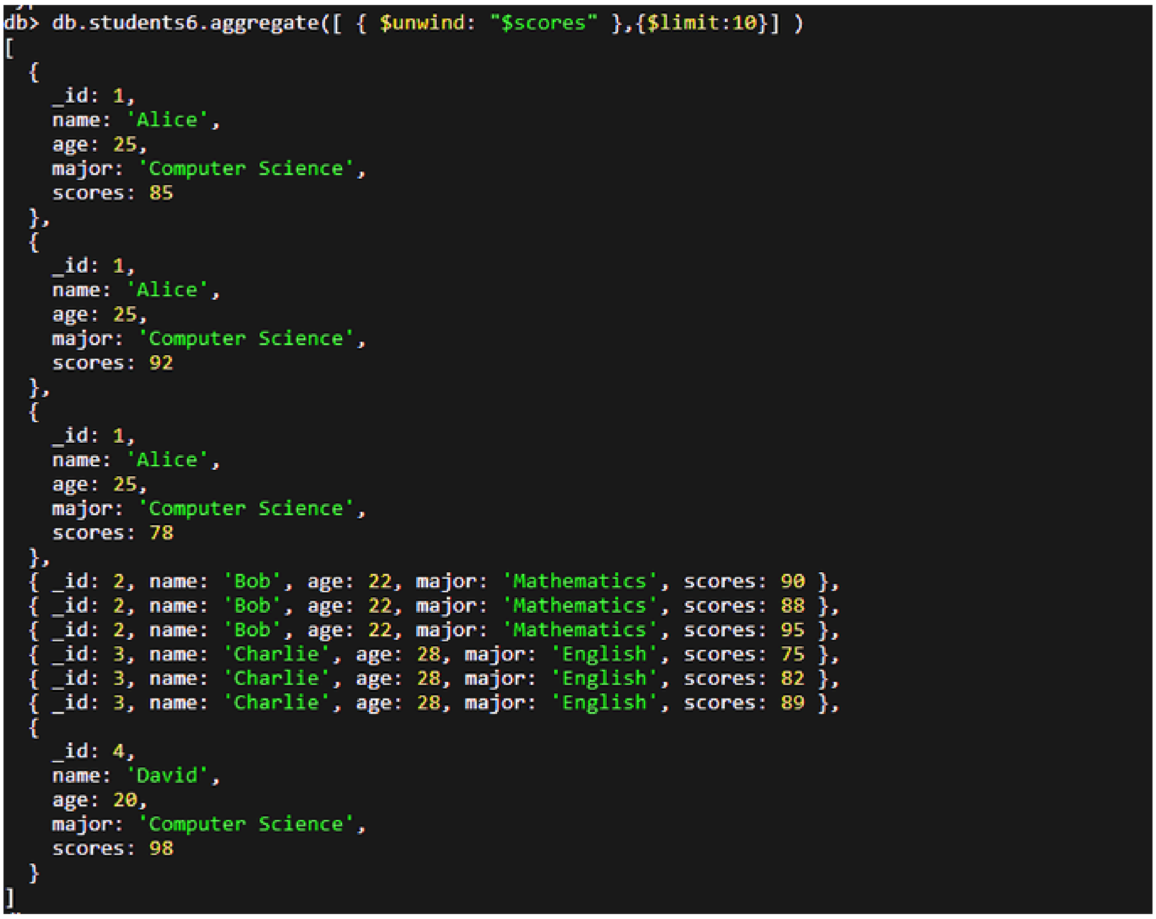


```
db> db.students6.aggregate([ { $limit: 5 }] )
[
  {
    _id: 1,
    name: 'Alice',
    age: 25,
    major: 'Computer Science',
    scores: [ 85, 92, 78 ]
  },
  {
    _id: 2,
    name: 'Bob',
    age: 22,
    major: 'Mathematics',
    scores: [ 90, 88, 95 ]
  },
  {
    _id: 3,
    name: 'Charlie',
    age: 28,
    major: 'English',
    scores: [ 75, 82, 89 ]
  },
  {
    _id: 4,
    name: 'David',
    age: 20,
    major: 'Computer Science',
    scores: [ 98, 95, 87 ]
  },
  {
    _id: 5,
    name: 'Eve',
    age: 23,
    major: 'Biology',
db>
  }
]
```

7. $UNWIND

The $unwind stage deconstructs an array field from the input documents to output a document for each element in the array. This is useful for working with documents that contain arrays and needing to process each element in the array as a separate document.
Example: Create a separate document for each score of each student.

syntax:db.students6.aggregate([
  { $unwind: "$scores" }
])



```
db> db.students6.aggregate([ { $unwind: "$scores" },{$limit:10}] )
[
  {
    _id: 1,
    name: 'Alice',
    age: 25,
    major: 'Computer Science',
    scores: 85
  },
  {
    _id: 1,
    name: 'Alice',
    age: 25,
    major: 'Computer Science',
    scores: 92
  },
  {
    _id: 1,
    name: 'Alice',
    age: 25,
    major: 'Computer Science',
    scores: 78
  },
  { _id: 2, name: 'Bob', age: 22, major: 'Mathematics', scores: 90 },
  { _id: 2, name: 'Bob', age: 22, major: 'Mathematics', scores: 88 },
  { _id: 2, name: 'Bob', age: 22, major: 'Mathematics', scores: 95 },
  { _id: 3, name: 'Charlie', age: 28, major: 'English', scores: 75 },
  { _id: 3, name: 'Charlie', age: 28, major: 'English', scores: 82 },
  { _id: 3, name: 'Charlie', age: 28, major: 'English', scores: 89 },
  {
    _id: 4,
    name: 'David',
    age: 20,
    major: 'Computer Science',
    scores: 98
  }
]
```

Explanation:

1. $unwind: Deconstructs the scores array field and creates a separate document for each score.
2. $limit: Limits the result to only 10 documents.

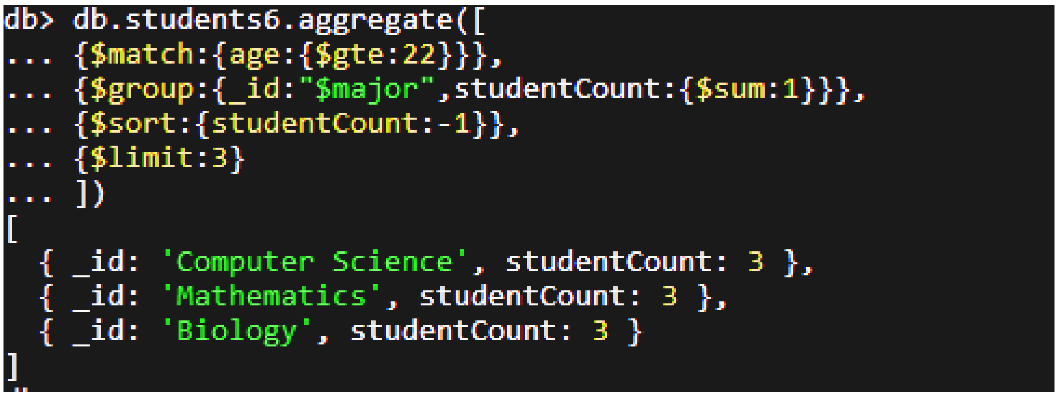
This will give you the first 10 documents that result from unwinding the scores array.

COMBINED EXAMPLE:

Let’s combine several of these stages to perform a more complex operation.

1.For instance, let's find the top 3 majors by the number of students, excluding those who are younger than 22, sorted by the number of students in descending order.

syntax:db.students6.aggregate([
  { $match: { age: { $gte: 22 } } }, // Step 1: Filter students who are at least 22 years old
  { $group: { _id: "$major", studentCount: { $sum: 1 } } }, // Step 2: Group by major and count students
  { $sort: { studentCount: -1 } }, // Step 3: Sort by student count in descending order
  { $limit: 3 } // Step 4: Limit to top 3 majors
])



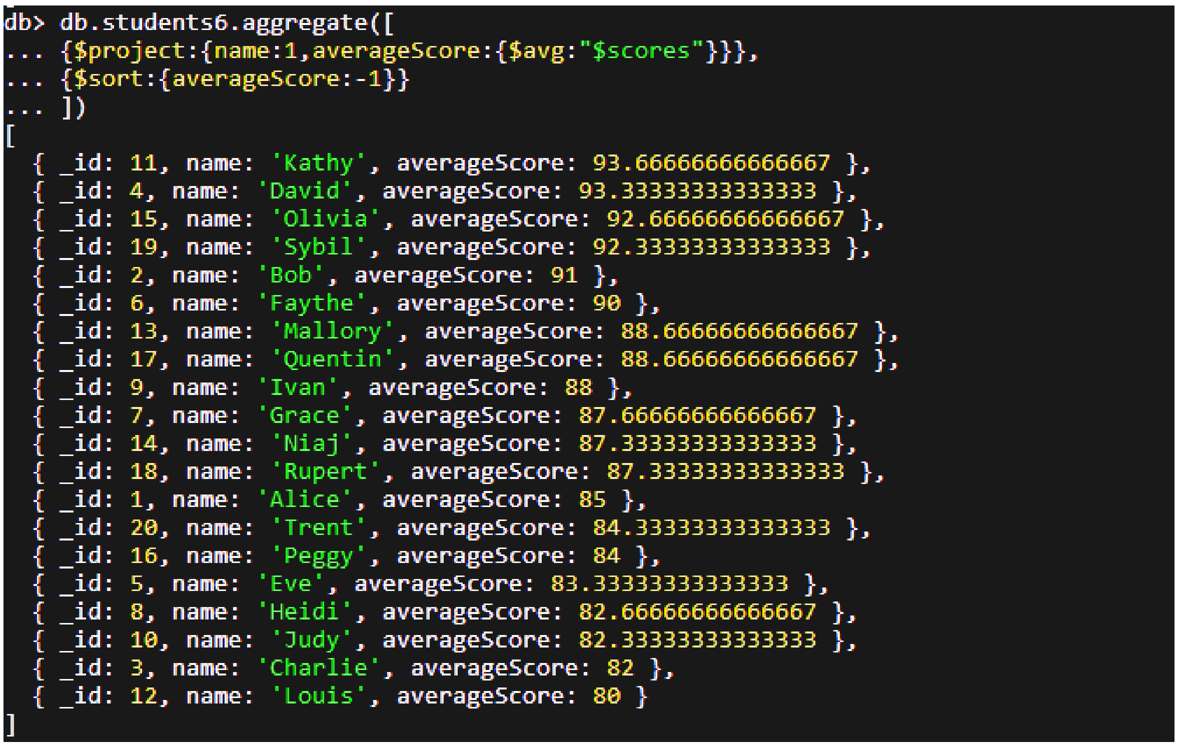
```
db> db.students6.aggregate([
... {$match:{age:{$gte:22}}},
... {$group:{_id:"$major",studentCount:{$sum:1}}},
... {$sort:{studentCount:-1}},
... {$limit:3}
... ])
[
  { _id: 'Computer Science', studentCount: 3 },
  { _id: 'Mathematics', studentCount: 3 },
  { _id: 'Biology', studentCount: 3 }
]
```

Explanation:

1. $match: Filters out students younger than 22.
2. $group: Groups the remaining students by their major and counts the number of students in each major.
3. $sort: Sorts the grouped documents by studentCount in descending order.
4. $limit: Limits the result to the top 3 majors.

2. Calculate Average Score per Student and Sort by Average Score:
Calculate the average score for each student and sort students by their average scores in descending order.

syntax:db.students6.aggregate([
  { $project: { name: 1, averageScore: { $avg: "$scores" } } }, // Step 1: Project name and calculate average score
  { $sort: { averageScore: -1 } } // Step 2: Sort by average score in descending order
])

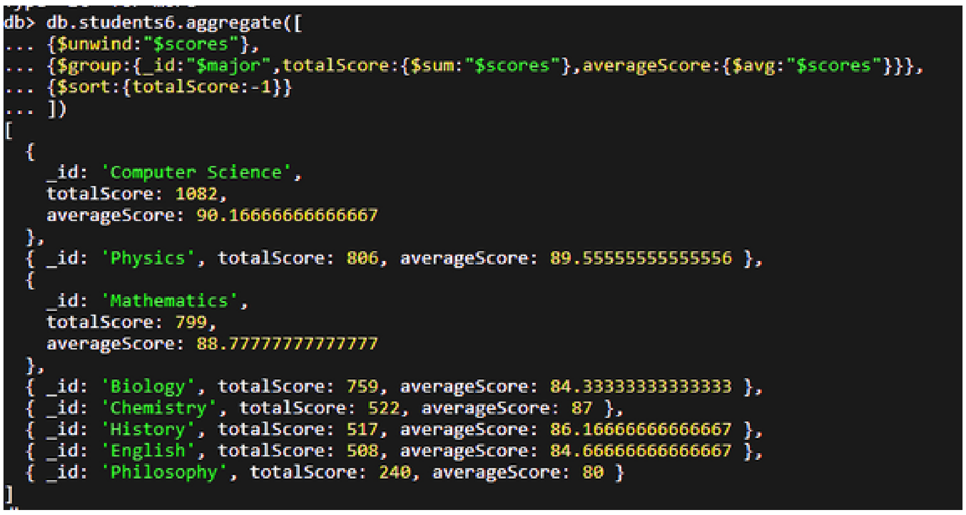


```
db> db.students6.aggregate([
... {$project:{name:1,averageScore:{$avg:"$scores"}}},
... {$sort:{averageScore:-1}}
... ])
[
  { _id: 11, name: 'Kathy', averageScore: 93.66666666666667 },
  { _id: 4, name: 'David', averageScore: 93.33333333333333 },
  { _id: 15, name: 'Olivia', averageScore: 92.66666666666667 },
  { _id: 19, name: 'Sybil', averageScore: 92.33333333333333 },
  { _id: 2, name: 'Bob', averageScore: 91 },
  { _id: 6, name: 'Faythe', averageScore: 90 },
  { _id: 13, name: 'Mallory', averageScore: 88.66666666666667 },
  { _id: 17, name: 'Quentin', averageScore: 88.66666666666667 },
  { _id: 9, name: 'Ivan', averageScore: 88 },
  { _id: 7, name: 'Grace', averageScore: 87.66666666666667 },
  { _id: 14, name: 'Niaj', averageScore: 87.33333333333333 },
  { _id: 18, name: 'Rupert', averageScore: 87.33333333333333 },
  { _id: 1, name: 'Alice', averageScore: 85 },
  { _id: 20, name: 'Trent', averageScore: 84.33333333333333 },
  { _id: 16, name: 'Peggy', averageScore: 84 },
  { _id: 5, name: 'Eve', averageScore: 83.33333333333333 },
  { _id: 8, name: 'Heidi', averageScore: 82.66666666666667 },
  { _id: 10, name: 'Judy', averageScore: 82.33333333333333 },
  { _id: 3, name: 'Charlie', averageScore: 82 },
  { _id: 12, name: 'Louis', averageScore: 80 }
]
```

Explanation:
1. $project: Includes the name field and calculates the average score for each student.
2. $sort: Sorts the documents by averageScore in descending order.

3. Group by Major and Calculate Total and Average Score:

Group students by their major, then calculate the total and average scores for each major.
syntax:db.students6.aggregate([
  { $unwind: "$scores" }, // Step 1: Unwind the scores array
    { $group: { _id: "$major", totalScore: { $sum: "$scores" }, averageScore: { $avg: "$scores" } } }, // Step 2: Group by major and calculate total and average scores
  { $sort: { totalScore: -1 } } // Step 3: Sort by total score in descending order
])



```
db> db.students6.aggregate([
... {$unwind:"$scores"},
... {$group:{_id:"$major",totalScore:{$sum:"$scores"},averageScore:{$avg:"$scores"}}},
... {$sort:{totalScore:-1}}
... ])
[
  {
    _id: 'Computer Science',
    totalScore: 1082,
    averageScore: 90.16666666666667
  },
  { _id: 'Physics', totalScore: 806, averageScore: 89.55555555555556 },
  {
    _id: 'Mathematics',
    totalScore: 799,
    averageScore: 88.77777777777777
  },
  { _id: 'Biology', totalScore: 759, averageScore: 84.33333333333333 },
  { _id: 'Chemistry', totalScore: 522, averageScore: 87 },
  { _id: 'History', totalScore: 517, averageScore: 86.16666666666667 },
  { _id: 'English', totalScore: 508, averageScore: 84.66666666666667 },
  { _id: 'Philosophy', totalScore: 240, averageScore: 80 }
]
```

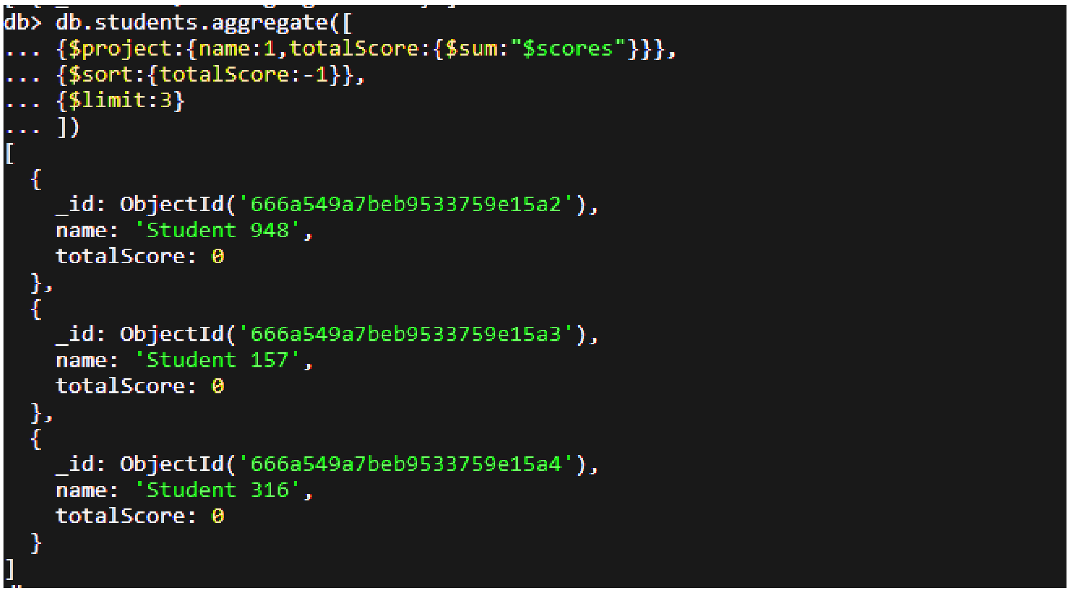
Explanation:

1. $unwind: Deconstructs the scores array field and creates a separate document for each score.
2. $group: Groups the documents by major and calculates the total and average scores for each major.
3. $sort: Sorts the documents by totalScore in descending order.

4. Find the Top 3 Students by Total Score:

Calculate the total score for each student and find the top 3 students by total score.

```
syntax:db.students6.aggregate([
  { $project: { name: 1, totalScore: { $sum: "$scores" } } },  // Step 1: Project name and calculate total score
  { $sort: { totalScore: -1 } },  // Step 2: Sort by total score in descending order
  { $limit: 3 }  // Step 3: Limit to top 3 students
])
```



```
db> db.students.aggregate([
... {$project:{name:1,totalScore:{$sum:"$scores"}}},
... {$sort:{totalScore:-1}},
... {$limit:3}
... ])
[
  {
    _id: ObjectId('666a549a7beb9533759e15a2'),
    name: 'Student 948',
    totalScore: 0
  },
  {
    _id: ObjectId('666a549a7beb9533759e15a3'),
    name: 'Student 157',
    totalScore: 0
  },
  {
    _id: ObjectId('666a549a7beb9533759e15a4'),
    name: 'Student 316',
    totalScore: 0
  }
]
```

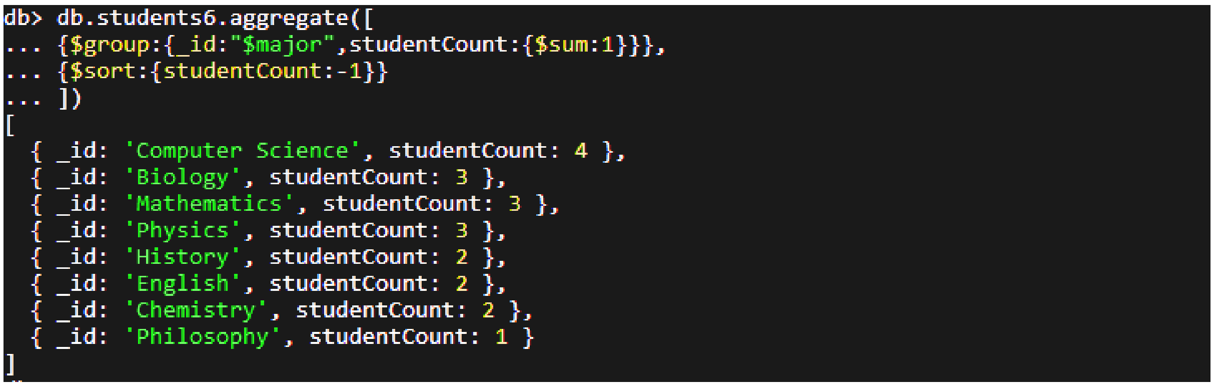
Explanation:

1. $project: Includes the name field and calculates the total score for each student.
2. $sort: Sorts the documents by totalScore in descending order.
3. $limit: Limits the result to the top 3 documents.

5. Calculate the Total Number of Students in Each Major:

Group students by their major and count the total number of students in each major.
syntax:

```
db.students6.aggregate([
  { $group: { _id: "$major", studentCount: { $sum: 1 } } },  // Step 1: Group by major and count students
  { $sort: { studentCount: -1 } }  // Step 2: Sort by student count in descending order
])
```



```
db> db.students6.aggregate([
... {$group:{_id:"$major",studentCount:{$sum:1}}},
... {$sort:{studentCount:-1}}
... ])
[
  { _id: 'Computer Science', studentCount: 4 },
  { _id: 'Biology', studentCount: 3 },
  { _id: 'Mathematics', studentCount: 3 },
  { _id: 'Physics', studentCount: 3 },
  { _id: 'History', studentCount: 2 },
  { _id: 'English', studentCount: 2 },
  { _id: 'Chemistry', studentCount: 2 },
  { _id: 'Philosophy', studentCount: 1 }
]
```

Explanation:

1. $group: Groups the documents by major and counts the number of students in each group.
2. $sort: Sorts the documents by studentCount in descending order.

These examples demonstrate how you can use various aggregation stages to perform complex data analysis tasks in MongoDB.