Devon Reing

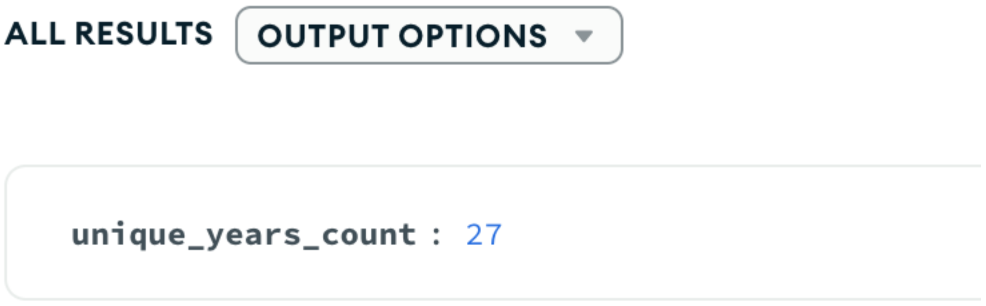
Dr. Forouraghi

Database Management Systems

22 November 2024

Homework 5

Each query below that I have written begins with db.unemployment.aggregate([…]). To avoid explaining the purpose of that portion of code for each piece, I will explain now it's purpose in each query below. The db portion is required in MongoDB and unemployment specifies to MongoDB that I want to access the unemployment database in the server I am currently running. Aggregate then allows me to apply multiple stages or filters on the documents within my unemployment database. Everything within the ([…]) helps to filter to list only the documents I wish to be included and grouped/calculated as needed.

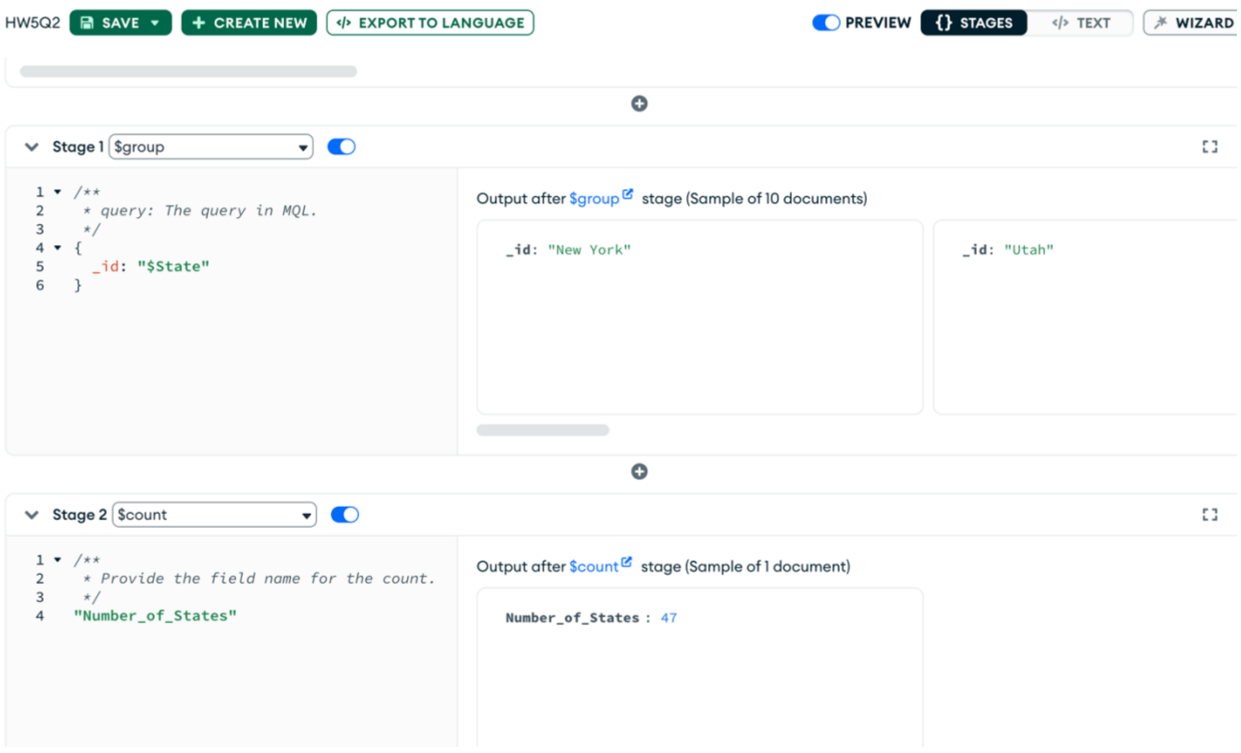


```
unique_years_count : 27
```

```
db.unemployment.aggregate([
 {
  $group: {
   _id: "$Year"
```

```
    }
  },
  {
    $count: "unique_years_count"
  }
])
```

The first query aims to find the number of unique years included in the unemployment data set. The group filter groups all documents with the same "Year" field value together and limits what is counted to be only the unique year values in the data set. The count stage then returns the number of years that are included from the group stage and returns that value to the user with the label "unique_years_count".



```
/**
 * query: The query in MQL.
 */
{
  _id: "$State"
}
```

Output after $group stage (Sample of 10 documents)

```
_id: "New York"
```

```
_id: "Utah"
```

```
/**
 * Provide the field name for the count.
 */
"Number_of_States"
```

Output after $count stage (Sample of 1 document)

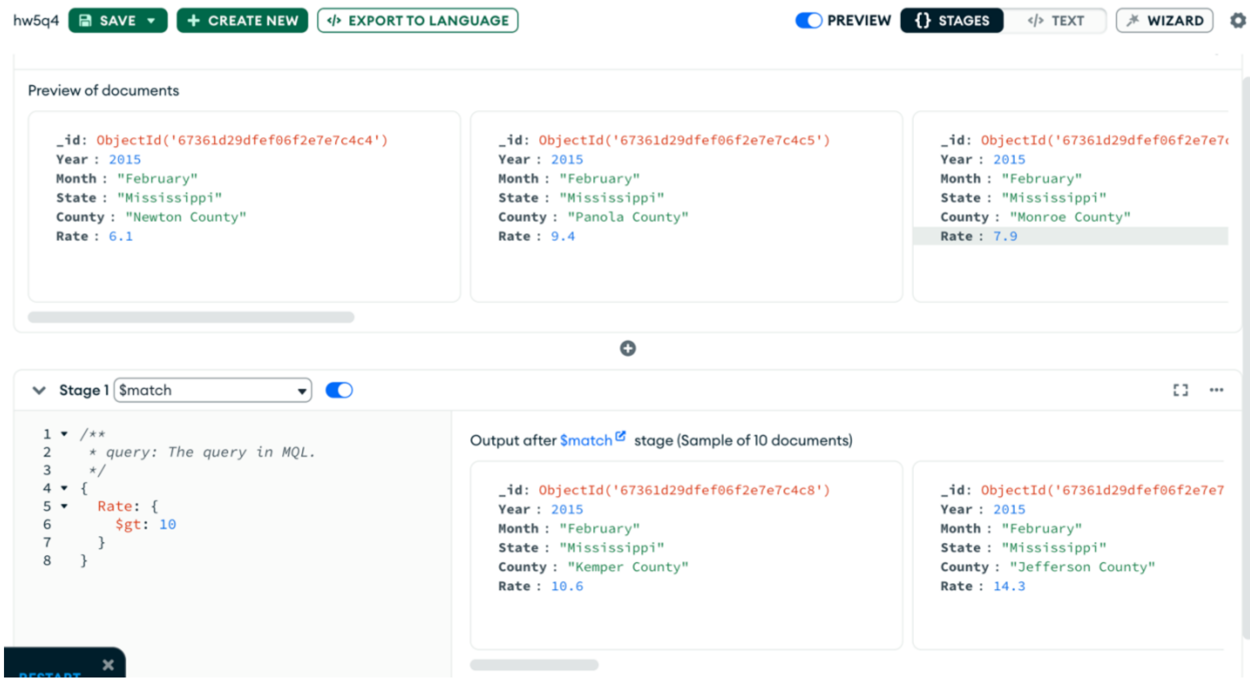
```
Number_of_States : 47
```

The second query aims to find the total number of states included in the data set. Similar to the first query, this works by first grouping the documents with the same value in the "State" field together. Once those documents are grouped and the id's associated with each group are listed using the group stage, the count stage then counts the number of unique id's listed and attaches the "Number_of_States" label to the output.

Question 3

db.unemployment.find({Rate : {$lt: 1.0}}).count()

This query given to us in the assignment finds the number of counties with a yearly unemployment rate < 1.0%. This does not specify that they need to be unique counties, so it is possible that the same county is counted twice if the county qualifies for two separate years.



Preview of documents

```
_id: ObjectId('67361d29dfef06f2e7e7c4c4')
Year : 2015
Month : "February"
State : "Mississippi"
County : "Newton County"
Rate : 6.1
```

```
_id: ObjectId('67361d29dfef06f2e7e7c4c5')
Year : 2015
Month : "February"
State : "Mississippi"
County : "Panola County"
Rate : 9.4
```

```
_id: ObjectId('67361d29dfef06f2e7e7c
Year : 2015
Month : "February"
State : "Mississippi"
County : "Monroe County"
Rate : 7.9
```

Stage 1 $match

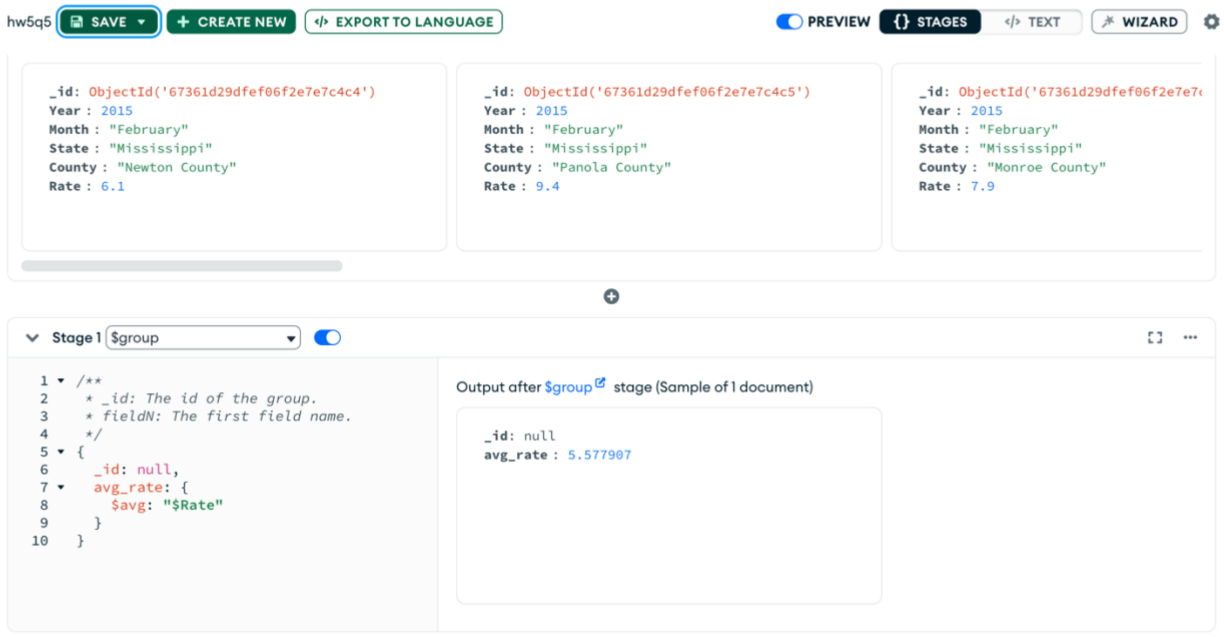
```
/**
 * query: The query in MQL.
 */
{
  Rate: {
    $gt: 10
  }
}
```

Output after $match stage (Sample of 10 documents)

```
_id: ObjectId('67361d29dfef06f2e7e7c4c8')
Year : 2015
Month : "February"
State : "Mississippi"
County : "Kemper County"
Rate : 10.6
```

```
_id: ObjectId('67361d29dfef06f2e7e7
Year : 2015
Month : "February"
State : "Mississippi"
County : "Jefferson County"
Rate : 14.3
```

The fourth query aims to find all counties with an unemployment rate greater than 10%. To achieve this, I used a match stage, which finds all the documents that fulfill the given criteria in the stage. The criteria I specified was for the “Rate” field in the document to contain a value that was greater than 10. Using $gt: 10 as the key in the key-value pair associated with the “Rate” field in the match stage ensures that only the documents with a value higher than 10 is included in the output list.



```
_id: ObjectId('67361d29dfef06f2e7e7c4c4')
Year : 2015
Month : "February"
State : "Mississippi"
County : "Newton County"
Rate : 6.1
```

```
_id: ObjectId('67361d29dfef06f2e7e7c4c5')
Year : 2015
Month : "February"
State : "Mississippi"
County : "Panola County"
Rate : 9.4
```

```
_id: ObjectId('67361d29dfef06f2e7e7c
Year : 2015
Month : "February"
State : "Mississippi"
County : "Monroe County"
Rate : 7.9
```

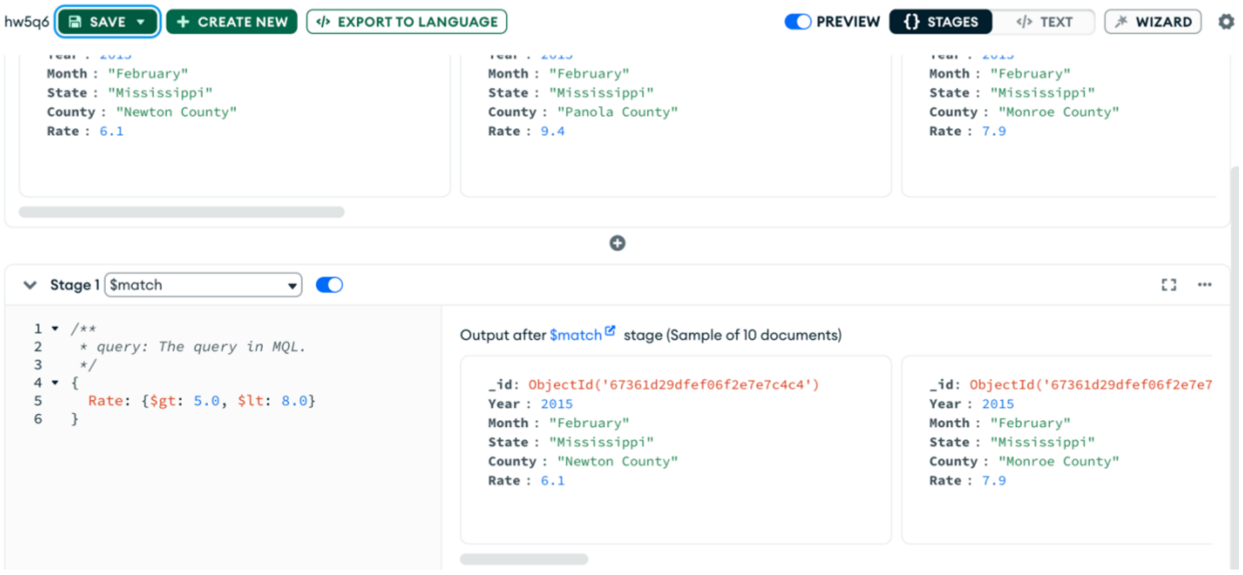
Stage 1 $group

```
/**
 * _id: The id of the group.
 * fieldN: The first field name.
 */
{
  _id: null,
  avg_rate: {
    $avg: "$Rate"
  }
}
```

Output after $group stage (Sample of 1 document)

```
_id: null
avg_rate : 5.577907
```

The fifth query aims to return the average unemployment rate across all states. To be able to calculate the average of every document in the list and therefore the average across all states, I used a group stage with an _id value of null. Whenever a group stage has an _id of null, all the documents are grouped together allowing for calculations including all the values. To calculate the average rate, I used the avg aggregate and specified that the “Rate” field should be used to calculate the average. Once that average was computed, it was then returned as the output with the “avg_rate” label attached.



```
Month : "February"
State : "Mississippi"
County : "Newton County"
Rate : 6.1

Month : "February"
State : "Mississippi"
County : "Panola County"
Rate : 9.4

Month : "February"
State : "Mississippi"
County : "Monroe County"
Rate : 7.9
```

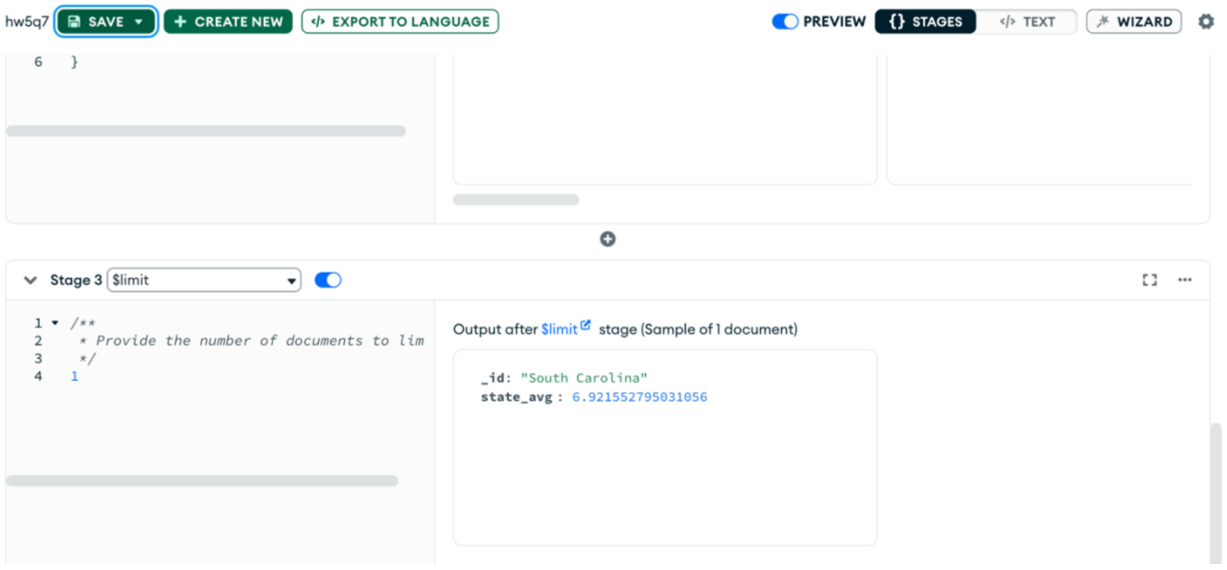
```
/**
 * query: The query in MQL.
 */
{
  Rate: {$gt: 5.0, $lt: 8.0}
}
```

Output after $match stage (Sample of 10 documents)

```
_id: ObjectId('67361d29dfef06f2e7e7c4c4')
Year : 2015
Month : "February"
State : "Mississippi"
County : "Newton County"
Rate : 6.1

_id: ObjectId('67361d29dfef06f2e7e7
Year : 2015
Month : "February"
State : "Mississippi"
County : "Monroe County"
Rate : 7.9
```

The sixth query aims to find all the counties with an unemployment rate between 5 and 8%. This is achieved through a match stage that filters all of the documents, and subsequently all the counties since every document is for a county for a specific year, by the “Rate” field. The $gt and $lt aggregate work together to form the range of values that the “Rate” field can fall between and act as a filter for all the documents. Once only the documents that fit the criteria of having an unemployment rate between 5 and 8% are selected, the documents are included in the output list.



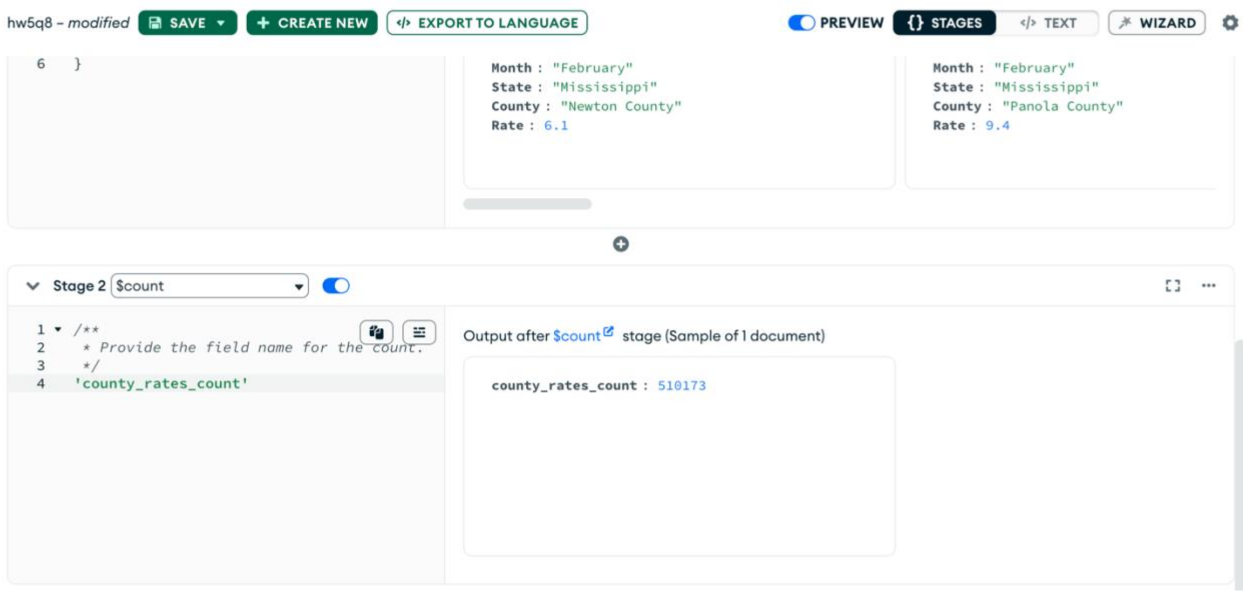
```
}
```

```
/**
 * Provide the number of documents to lim
 */
1
```

Output after $limit stage (Sample of 1 document)

```
_id: "South Carolina"
state_avg : 6.921552795031056
```

The seventh query aims to find the state with the highest unemployment rate. Since the documents refer to counties and not states, to find the state with the highest unemployment rate we need to find the average unemployment rate of each state. To achieve this, a group stage is first used to group the documents with the same value in the “State” field together through the _id field portion of the stage. The second portion of the group stage then takes the groups of each state and performs an average by using the $avg aggregate on the “Rate” field. The result of these averages is then labeled “state_avg”. Once that group stage is complete, the results of the group stage are sorted in descending order indicted by the 1 in the sort stage. Finally, to get only the highest value, a limit stage is used and the 1 in the stage indictaes that only 1 value from the top of the list should be returned, hence returning the state with the highest average unemployment rate.



```
}
```

```
Month : "February"
State : "Mississippi"
County : "Newton County"
Rate : 6.1
```

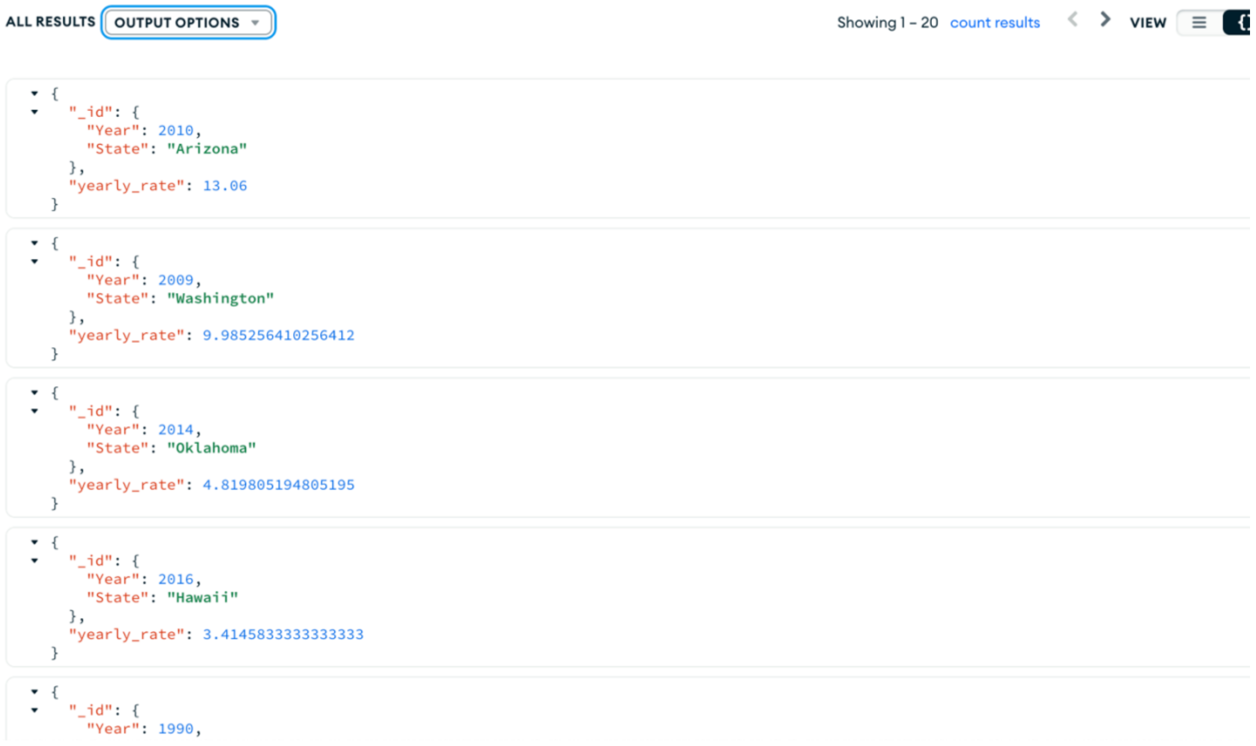
```
Month : "February"
State : "Mississippi"
County : "Panola County"
Rate : 9.4
```

```
/**
 * Provide the field name for the count.
 */
'county_rates_count'
```

Output after $count stage (Sample of 1 document)

```
county_rates_count : 510173
```

The eighth query aims to count how many counties have an unemployment rate above 5%. To achieve this, I first used a match stage to find all the documents where the value of the “Rate” field was above 5% using the aggregate $gt. This returns all the documents that match the criteria of having an unemployment rate of over 5%, and subsequently lists all the counties that fulfill the criteria since each document refers to a county in a specific year. To get the total count of the number of documents in the list, I then used a count stage which grabs the output list generated in the stage above and counts the number of documents in the list. I then also used the count stage to attach a label of "county_rates_count" to the output generated that lists the count.



```
{
  "_id": {
    "Year": 2010,
    "State": "Arizona"
  },
  "yearly_rate": 13.06
}
{
  "_id": {
    "Year": 2009,
    "State": "Washington"
  },
  "yearly_rate": 9.985256410256412
}
{
  "_id": {
    "Year": 2014,
    "State": "Oklahoma"
  },
  "yearly_rate": 4.819805194805195
}
{
  "_id": {
    "Year": 2016,
    "State": "Hawaii"
  },
  "yearly_rate": 3.4145833333333333
}
{
  "_id": {
    "Year": 1990,
```

The ninth query aims to calculate the average unemployment rate per state by year. To do this, I first used a group stage and utilized composite key-value pairs to group the documents to be per state and per year. In the _id field of the group stage, I used braces to create a list of fields to group the documents by. In the brackets, I listed key-value pairs for both the "State" and "Year" fields which separated the documents as needed to then calculate the average for each group created. After the _id portion of the group stage, I then used the $avg aggregate to calculate the average of the values in the "Rate" field for each group and attached the "yearly_rate" label to this calculated value. The output generated then included an array object listing the state and year the yearly rate listed below is associated with.