

- **Title:** DB Assignment 5
- **Your Name:** Deare
- **Date:** 2024/11/11

Step1:

- Created the collection **unemployment** in the **test** database and populated it with the downloaded CSV data.

Step 2:

- Import Year and **Rate** fields as **Number** type, Month, State and Country fields as **String** type.

**Import**

To collection test.unemployment

Import file: unemployment\_US.csv

**Options**

Select delimiter: Comma

☒ Ignore empty strings

☐ Stop on errors

**Specify Fields and Types** [Learn more about data types](#)

	Year	Month	State	County	Rate
	Number	String	String	String	Number
1	2015	February	Mississippi	Newton County	6.1
2	2015	February	Mississippi	Panola County	9.4
3	2015	February	Mississippi	Monroe County	7.9
4	2015	February	Mississippi	Hinds County	6.1

Cancel Import

- Successfully imported 885548 documents

MongoDB Compass - myMongo/test/unemployment

Connections Edit View Collection Help

Compass

myMongo > test > unemployment

Documents 885.6K Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

EXPLAIN RESET FIND Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 25 885548

```
{
  "_id": ObjectId("6732547f668b4718f704cb1d"),
  "Year": 2015,
  "Month": "February",
  "State": "Mississippi",
  "County": "Newton County",
  "Rate": 6.1
}
```

```
{
  "_id": ObjectId("6732547f668b4718f704cb1e"),
  "Year": 2015,
  "Month": "February",
  "State": "Mississippi",
  "County": "Panola County",
  "Rate": 9.4
}
```

```
{
  "_id": ObjectId("6732547f668b4718f704cb1f"),
  "Year": 2015,
  "Month": "February",
  "State": "Mississippi",
  "County": "Monroe County",
  "Rate": 7.9
}
```

Import completed. 885548 documents imported.

Step 3:

■ Analyze the schema:

- Click on the 'Schema' tab and select 'Analyze Schema' to view the results of the schema analysis.
- **Collection Name:** unemployment
- **Sample Size:** The analysis is based on a sample of 1000 documents.

Field Analysis:

1. \_id (ObjectId)

**Description:** This field automatically generated by MongoDB serves as the **primary key for the collection**. It uniquely identifies each document in the collection.

2. County (String)

**Distribution:** The graph shows a variety of counties represented with varying frequencies.

**Analysis:** The distribution suggests a balanced representation of data across different counties with 1% frequencies and Chariton Country, Mahaska Country, and Jackson Country have the highest 2% frequencies which is useful for regional analysis.

3. Month (String)

**Distribution:** Months are represented with varying frequencies.

**Analysis:** January has height frequencies with a percentage of 15%, and September has the lowest frequencies with a percentage of 4%.

4. Rate (Double)

**Distribution:** This field shows a rate and with its percentage, likely represents unemployment rates.

**Analysis:** *Under min and max for Rate:* Min: 0.7 - This represents the lowest value found in the Rate field across the sampled documents. Max: 34.6 - This represents the highest value found in the Rate field.

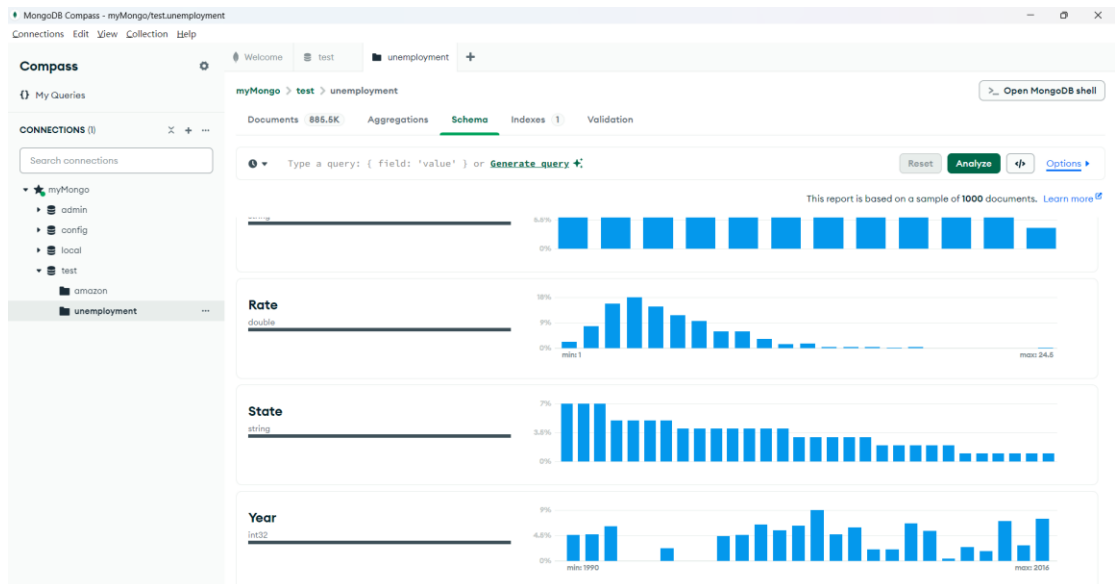
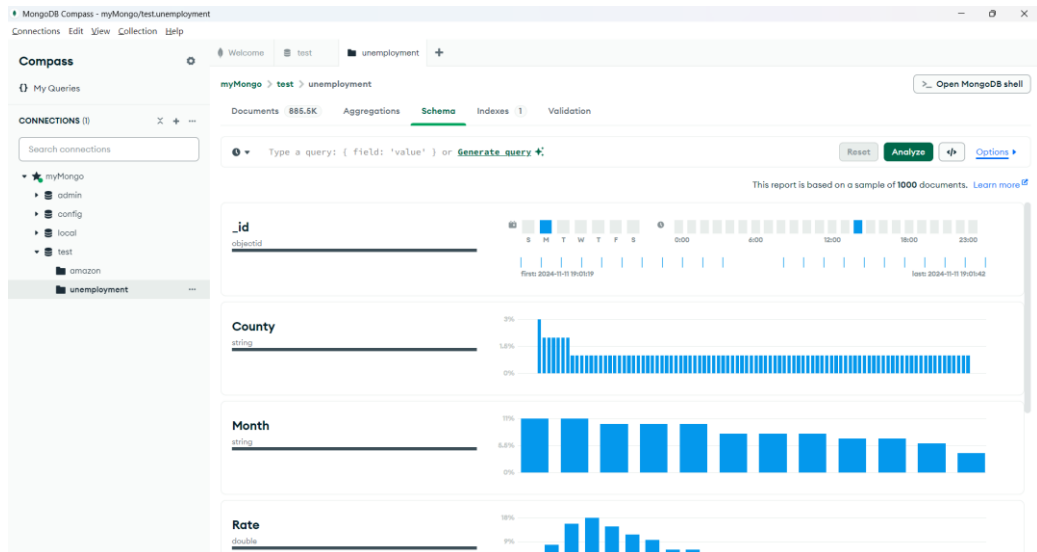
5. State (String)

**Analysis:** Graph distribution indicates that the data encompasses a variety of states. Missouri has the highest frequency with a percentage of 8%, "New Mexico", "Wyoming", "Washington", "California", "Rhode Island", "New York" and "Utah" has the lowest frequency with a percentage of 1%.

6. Year (Int32)

**Range:** Min: 1990, Max: 2015

**Analysis:** Reflecting the years from 1990 to 2015 across the sampled documents. 2010-2011 has the highest frequencies with percentage of 7%. Year fields could be useful for trend analysis over time.



Step 4:

Queries in **MongoDB Shell**:

### 1. Over how many years was the unemployment data collected?

Means: the total span of years during which the unemployment data was gathered.

This involves determining the range of years covered in the dataset.

I am going to use aggregation group by year fields and use count to count year spans.

```
db.unemployment.aggregate([
```

```
{
```

```
  $group:
```

```
    /**
```

```
      * _id: The id of the group.
```

```
      * fieldN: The first field name.
```

```
    */
```

```

    {
      _id: "$Year"
    }
  },
  {
    $count:
      /**
       * Provide the field name for the count.
       */
      "YearSpan"
  });

```

localhost:27017 > test > unemployment

Documents 885.5K Aggregations Schema Indexes 1 Validation

\$group

\$count

Edit

ALL RESULTS
OUTPUT OPTIONS

YearSpan : 27

]

Data on unemployment rates were collected over 27 years.

## 2. How many states were reported on in this dataset?

Assuming its meaning is that this question asked for a distinct state name and its total number.

A solution similar to question 1, this time going to group by State fields and count the state in an aggregate query.

```

db.getCollection('unemployment').aggregate(
  [
    { $group: { _id: '$State' } },
    { $count: 'NumberOfState' }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);

```

The screenshot shows the MongoDB Atlas Aggregation Pipeline Builder interface. It displays two stages of an aggregation pipeline:

- Stage 1 (\$group):** The query is:
 

```
1 /**
2  * _id: The id of the group.
3  * fieldN: The first field name.
4  */
5 {
6   _id: "$State"
7 }
```

 The output shows a sample of 10 documents with two visible entries: `{ "_id": "Rhode Island" }` and `{ "_id": "Ohio" }`.
- Stage 2 (\$count):** The query is:
 

```
1 /**
2  * Provide the field name for the count.
3  */
4 "NumberOfState"
```

 The output shows a sample of 1 document: `{ "NumberOfState": 47 }`.

3. What does this query compute?

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

This query means that retrieved unemployment people who have a Rate less than 1.0 and then count their number.

```
> db.unemployment.find({Rate: {$lt: 1.0}}).count()
< 657
```

4. Find all **counties** with unemployment rate higher than **10%**

Assuming this question asks for a filter for states with an unemployment rate higher than 10%, which I think means retrieving values greater than 10.0 in the Rate field.

```
db.getCollection('unemployment').aggregate(
[
  {
    $project: {
      _id: 0,
      State: 1,
      County: 1,
      Rate: 1
    }
  },
  { $match: { Rate: { $gt: 10 } } }
],
{ maxTimeMS: 60000, allowDiskUse: true }
);
```

```

1 [
2   {
3     $project:
4     /**
5      * specifications: The fields to
6      * include or exclude.
7      */
8     {
9       _id: 0,
10      State: 1,
11      County: 1,
12      Rate: 1
13    },
14  },
15  {
16    $match:
17    /**
18     * query: The query in MQL.
19     */
20    {
21      Rate: {
22        $gt: 10.0
23      }
24    }
25  }
26 ]

```

**PIPELINE OUTPUT**  
 Sample of 10 documents
 

State : "Mississippi"  
 County : "Kemper County"  
 Rate : 10.6

State : "Mississippi"  
 County : "Jefferson County"  
 Rate : 14.3

State : "Mississippi"  
 County : "Sharkey County"  
 Rate : 11.1

State : "Mississippi"  
 County : "Tunica County"  
 Rate : 11.5

State : "Mississippi"  
 County : "Noxubee County"  
 Rate : 10.6

Project

Smatch

Edit

?

Explain

Export

Run

Options

**ALL RESULTS**

OUTPUT OPTIONS

Showing 1 – 20 count results

VIEW

State : "Mississippi"  
 County : "Kemper County"  
 Rate : 10.6

State : "Mississippi"  
 County : "Jefferson County"  
 Rate : 14.3

State : "Mississippi"  
 County : "Sharkey County"  
 Rate : 11.1

State : "Mississippi"  
 County : "Tunica County"  
 Rate : 11.5

State : "Mississippi"  
 County : "Noxubee County"  
 Rate : 10.6

State : "Mississippi"  
 County : "Sunflower County"  
 Rate : 10.6

## 5. Calculate the average unemployment rate across all states.

Assuming the result includes all the state names and its **average unemployment rates**.

```

db.getCollection('unemployment').aggregate(
[
  { $project: { _id: 0, State: 1, Rate: 1 } },
  {
    $group: {
      _id: '$State',
      averageRATE: { $avg: '$Rate' }
    }
  }
],
{ maxTimeMS: 60000, allowDiskUse: true }
);

```

This is sample size's result:

Untitled - modified **SAVE** **CREATE NEW** **EXPORT TO LANGUAGE** **PREVIEW** **STAGES** **TEXT** **WIZARD**

```

1 [
2   {
3     $project:
4       /**
5        * specifications: The fields to
6        * include or exclude.
7        */
8     {
9       _id: 0,
10      State: 1,
11      Rate: 1
12    },
13  },
14  {
15    $group:
16      /**
17       * _id: The id of the group.
18       * fieldN: The first field name.
19       */
20    {
21      _id: "$State",
22      averageRATE: {
23        $avg: "$Rate"
24      }
25    }
26  }
27 ]

```

**PIPELINE OUTPUT**  
Sample of 10 documents

**OUTPUT OPTIONS**

_id: "New Hampshire"	averageRATE : 3.4988571428571427
_id: "Utah"	averageRATE : 4.855960591133005
_id: "New York"	averageRATE : 5.655852534562213
_id: "New Jersey"	averageRATE : 6.206802721088436
_id: "Hawaii"	averageRATE : 4.058571428571429
_id: "Ohio"	averageRATE : 5.659772727272727

This is after clicking run button to get all the result:

**ALL RESULTS** **OUTPUT OPTIONS** Showing 1 - 20 count results **VIEW**

_id: "New Hampshire"	averageRATE : 4.34570987654321
_id: "Utah"	averageRATE : 5.503075776926352
_id: "New York"	averageRATE : 6.226224611708482
_id: "New Jersey"	averageRATE : 6.422104644326867
_id: "Hawaii"	averageRATE : 5.730401234567902
_id: "Ohio"	averageRATE : 6.923390151515152
_id: "Minnesota"	averageRATE : 5.403749822619554

## 6. Find all counties with an unemployment rate between 5% and 8%.

I assume this question is asked about all state names and county names where the unemployment rate values between 5.0 and 8.0.

```

db.getCollection('unemployment').aggregate(
[
  { $match: { Rate: { $gte: 5, $lte: 8 } } },
  {
    $project: {
      _id: 0,
      State: 1,
      County: 1,
      Rate: 1
    }
  }
],
{ maxTimeMS: 60000, allowDiskUse: true }
);

```

```

1 [
2   {
3     $match:
4     /**
5      * query: The query in MQL.
6      */
7     {
8       Rate: {
9         $gte: 5,
10        $lte: 8
11      }
12    },
13  },
14  {
15    $project:
16    /**
17     * specifications: The fields to
18     * include or exclude.
19     */
20    {
21      _id: 0,
22      State: 1,
23      County: 1,
24      Rate: 1
25    }
26  }
27 ]

```

PIPELINE OUTPUT

Sample of 10 documents

State : "Mississippi"

County : "Newton County"

Rate : 6.1

State : "Mississippi"

County : "Monroe County"

Rate : 7.9

State : "Mississippi"

County : "Hinds County"

Rate : 6.1

State : "Mississippi"

County : "Calhoun County"

Rate : 6.9

State : "Mississippi"

County : "Clarke County"

Rate : 7.9

State : "Mississippi"

County : "Newton County"

Rate : 6.1

State : "Mississippi"

County : "Monroe County"

Rate : 7.9

State : "Mississippi"

County : "Hinds County"

Rate : 6.1

State : "Mississippi"

County : "Calhoun County"

Rate : 6.9

State : "Mississippi"

County : "Clarke County"

Rate : 7.9

State : "Mississippi"

County : "Tate County"

Rate : 7.9

7. Find the state with the **highest unemployment rate**. Hint. Use { \$limit: 1 }
- ```

db.getCollection('unemployment').aggregate(
[
  { $sort: { Rate: -1 } },
  { $limit: 1 },
  { $project: { _id: 0, State: 1, Rate: 1 } }
],
{ maxTimeMS: 60000, allowDiskUse: true });

```

Untitled - modified

SAVE

CREATE NEW

EXPORT TO LANGUAGE

PREVIEW

STAGES

TEXT

WIZARD

```

1 [
2   {
3     $sort:
4     /**
5      * Provide any number of field/order pairs.
6      */
7     {
8       Rate: -1
9     }
10  },
11  {
12    $limit:
13    /**
14     * Provide the number of documents to limit.
15     */
16    1
17  },
18  {
19    $project:
20    {
21      _id: 0,
22      State: 1,
23      Rate: 1
24    }
25  }
26 ]

```

PIPELINE OUTPUT

Sample of 1 document

State : "Colorado"

Rate : 58.4



8. Count how many counties have an unemployment rate above **5%**.  
Assuming this question is about asking unemployment Rate value greater than 5.0.

The screenshot shows the MongoDB Aggregation Pipeline Builder interface. The pipeline is defined as follows:

```
1 [
2   {
3     $match: {
4       Rate: {
5         $gt: 5.0
6       }
7     },
8   },
9   {
10    $group: {
11      _id: "$County" // Group by 'County' to ensure each county is counted
12    },
13  },
14  {
15    $count:
16      /**
17       * Provide the field name for the count.
18       */
19    "NumberOfCounties"
20  }
21 ]
```

The pipeline output shows a sample of 1 document:

```
NumberOfCounties : 1697
```

The interface includes buttons for **SAVE**, **CREATE NEW**, **EXPORT TO LANGUAGE**, **PREVIEW**, **STAGES**, **TEXT**, **WIZARD**, and **OUTPUT OPTIONS**. The **Run** button is highlighted in green. The results section shows **ALL RESULTS** and **OUTPUT OPTIONS** tabs, with a message "Showing 1 - 1 count results". The result is displayed as:

```
NumberOfCounties : 1736
```

```
db.getCollection('unemployment').aggregate(
[
  { $match: { Rate: { $gt: 5 } } },
  { $group: { _id: '$County' } },
  { $count: 'NumberOfCounties' }
],
{ maxTimeMS: 60000, allowDiskUse: true }
);
```

9. Calculate the average unemployment rate per state by year.  
The aggregation pipeline would involve grouping the data first by state and year, calculating the average rate for these groups, and then projecting the results in a readable format for example I will sort by state and then year for easier reading.

```
db.getCollection('unemployment').aggregate(
[
  {
    $group: {
      _id: { State: '$State', Year: '$Year' },
      countAverageRate: { $avg: '$Rate' }
    }
  },
],
```

```

{
  $project: {
    _id: 0,
    State: '$_id.State', // Map 'State' from the group '_id'
    Year: '$_id.Year', // Map 'Year' from the group '_id'
    countAverageRate: 1
  }
},
{ $sort: { State: 1, Year: 1 } }
],
{ maxTimeMS: 60000, allowDiskUse: true }
);

```

The screenshot shows a MongoDB aggregation pipeline in a web interface. The left pane displays the query, and the right pane shows the pipeline output.

**Query:**

```

1 {
2   $group: {
3     _id: {
4       State: "$State",
5       Year: "$Year"
6     },
7     countAverageRate: {
8       $avg: "$Rate"
9     }
10  },
11  $project: {
12    _id: 0,
13    State: "$_id.State",
14    Year: "$_id.Year",
15    countAverageRate: 1
16  }
17 },
18 { $sort: {
19   State: 1,
20   Year: 1
21 } }
22 ]

```

**PIPELINE OUTPUT**  
Sample of 10 documents

| countAverageRate  | State   | Year |
|-------------------|---------|------|
| 7.917164179104478 | Alabama | 2014 |
| 7.033084577114428 | Alabama | 2015 |
| 6.806784260515604 | Alabama | 2016 |
| 9.478555555555556 | Arizona | 2014 |
| 8.791666666666666 | Arizona | 2015 |

10. (Extra Credit) For each state, calculate the total unemployment rate across all counties (sum of all county rates)

Use an aggregation pipeline that groups the data by state, sums the unemployment rates for all counties within each state, and then projects the results in a readable format, and then sorts the result by state name in alphabetical order, which is not a requirement, however, I want to add this extra step for my result more readable.

```
db.getCollection('unemployment').aggregate(
```

```

[
  {
    $group: {
      _id: '$County',
      TOTALsumRate: { $sum: '$Rate' }
    }
  },
  {
    $project: {
      _id: 0,
      County: '$_id',

```

```

        TOTALsumRate: 1
      }
    },
    { $sort: { State: 1 } }
  ],
  { maxTimeMS: 60000, allowDiskUse: true }
);

```

The screenshot displays a MongoDB Aggregation Pipeline interface. On the left, a query is written in a code editor, showing a pipeline with a \$group stage and a \$sort stage. On the right, the 'PIPELINE OUTPUT' section shows a sample of 10 documents. Each document contains 'TOTALsumRate' and 'County' fields. Below this, a scrollable list shows more documents with the same structure.

**Query:**

```

1 [
2   {
3     $group: {
4       _id: "$County",
5       TOTALsumRate: {
6         $sum: "$Rate"
7       }
8     },
9   },
10  {
11    $project: {
12      _id: 0,
13      County: "$_id",
14      TOTALsumRate: 1
15    },
16  },
17  {
18    $sort: {
19      State: 1
20    }
21  }
22 ]

```

**PIPELINE OUTPUT**  
Sample of 10 documents

| TOTALsumRate | County                  |
|--------------|-------------------------|
| 159.5        | "Monona County"         |
| 281.8        | "Unicoi County"         |
| 158.4        | "Warrick County"        |
| 111.3        | "Ida County"            |
| 194.2        | "Cannon County"         |
| 153.8        | "Henrico County"        |
| 4815.4       | "Northumberland County" |
| 1941.1       | "Woodson County"        |
| 1548.1       | "Warrick County"        |
| 2893         | "Yazoo County"          |
| 3382.1       | "Lane County"           |
| 1528.3       | "Monona County"         |
| 4341.4       | "Bristol County"        |

11. (Extra Credit) The same as Query 10 but for states with data from 2015 onward  
The output from this pipeline will be documents where each document represents a state, along with the total summed unemployment rate of all its counties from 2015 onward, and then the result by state name in alphabetical order, which is not a requirement, however, I want to add this extra step for my result more readable.

```

db.getCollection('unemployment').aggregate(
[
  { $match: { Year: { $gt: 2015 } } },
  {
    $group: {
      _id: '$State',

```

```

        TOTALsumRateAFTER2015: { $sum: '$Rate' }
    }
},
{
    $project: {
        _id: 0,
        State: '$_id',
        TOTALsumRateAFTER2015: 1
    }
},
{ $sort: { State: 1 } }
],
{ maxTimeMS: 60000, allowDiskUse: true }
);

```

```

1  {
2  {
3  {
4  { $match:
5  {
6  { Year: {
7  { $gt: 2015
8  }
9  }
10 } },
11 { $group:
12 {
13 {
14 { _id: "$State",
15   TOTALsumRateAFTER2015: {
16   $sum: "$Rate"
17   }
18 } },
19 { $project:
20 {
21 {
22 { _id: 0,
23   State: "$_id",
24   TOTALsumRateAFTER2015: 1
25 }
26 } },
27 { $sort:
28 {
29 { State: 1
30 }

```

PIPELINE OUTPUT

Sample of 10 documents

TOTALsumRateAFTER2015 : 5446.1  
State : "Alabama"

TOTALsumRateAFTER2015 : 1391.9  
State : "Arizona"

TOTALsumRateAFTER2015 : 4191  
State : "Arkansas"

TOTALsumRateAFTER2015 : 4415.1  
State : "California"

TOTALsumRateAFTER2015 : 2623.6  
State : "Colorado"

TOTALsumRateAFTER2015 : 492.3  
State : "Connecticut"