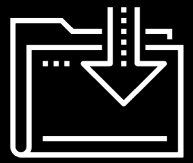
Aggregation, Analysis, and Integration with MongoDB

Data Boot Camp

Lesson 12.3




WELCOME

Class Objectives

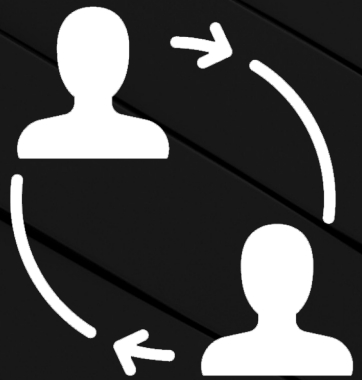
By the end of today's class, you will be able to:

Use aggregation and aggregation pipelines with MongoDB to analyze a subset of a MongoDB collection.

Convert a MongoDB result to a Pandas DataFrame.

Import data from an API to save to MongoDB.

Use data from a Mongo database to plot charts with Matplotlib.



PyMongo Warm-Up

Suggested Time:

15 minutes

PyMongo Warm-Up

Import the Data:

01

In your Terminal, navigate to the folder where your artifacts.json file is stored.

02

Import the dataset

```
mongoimport --type json -d met -c artifacts --drop --jsonArray
artifacts.json
```

PyMongo Warm-Up

Set up PyMongo:

03

Within Jupyter, open WarmUp_Unsolved.ipynb and use the following instructions to check that you created the database correctly, and set your variables:

- Import MongoClient from pymongo and pprint from pprint.
- Create an instance of MongoClient.
- Confirm that you imported the data correctly by listing your database names.
- Assign the met database to a variable name.
- Review the collections in your new database.
- Use `pprint` and `find_one()` to review a document in the artifacts collection.
- Assign the artifacts collection to a variable.

PyMongo Warm-Up

Review PyMongo Methods:

04

Review the code we have explored previously to find:

- How many documents have the `culture` "Nayarit" using `count_documents()`.
- How many documents have a height of at least 40cm using `$gte`.

05

Pretty print the results of a query that:

- Finds the documents where:
  - The culture is "Nayarit" or "Central American Isthmus" and
  - the height is less than or equal to 40cm
- Returns only the following fields: "title", "department", "culture", "measurements", and "objectURL"
- Sorts alphabetically by "title"
- Limits the results to 5

Aggregation in MongoDB and PyMongo

Aggregation Accumulators

Operator	Function
`$avg`	Returns an average of the specified expression or list of expressions for each document. Ignores non-numeric values.
`$sum`	Returns a sum of numerical values. Ignores non-numeric values.
`$max`	Returns the maximum of the specified expression or list of expressions for each document.
`$min`	Returns the minimum of the specified expression or list of expressions for each document.
`$stdDevPop`	Returns the population standard deviation of the input values.
`$stdDevSamp`	Returns the sample standard deviation of the input values.

Formatting Aggregation Accumulator Operations

```
query = [
          { '$group':
              { '_id': '$field_name_to_group_by',
                'averagePrice':
                    { '$avg', '$field_name_to_avg' }
              }
          }
        ]
```

The `$group` operator is used when performing aggregation accumulation operations.

The `'_id'` field is required and must reference the field(s) to group by.

Subsequent fields should be named to reference the aggregation accumulation to be performed.

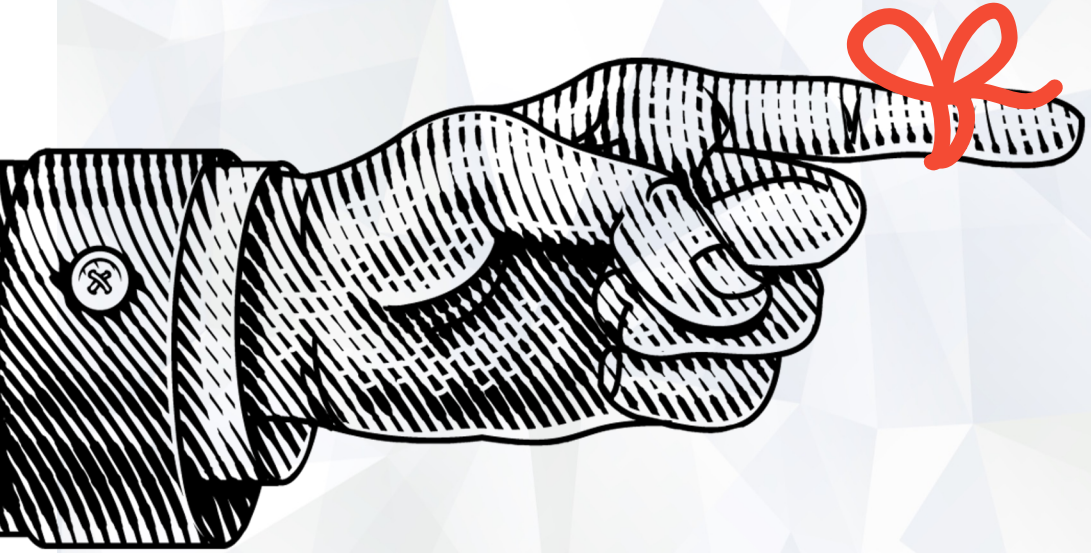
Formatting Aggregation Accumulator Operations

```
query = [
        { '$group':
          { '_id': '$field_name_to_group_by',
            'averagePrice':
              { '$avg', '$field_name_to_avg' }
          }
        }
]
```

Aggregation operations must be performed within a list.

To reference the field values, the field name must be preceded by a dollar sign '$'.

Aggregation accumulation operators follow the new field names we create.



Remember,

To reference field values, the field name must be preceded by a dollar sign:

```
$field_name
```

Instructor Demonstration

Aggregate by Country

Questions?

Activity: Aggregate by Classification

In this activity, you will practice using the aggregate method in MongoDB.

Suggested Time:

10 Minutes

Activity: Aggregate by Classification

Instructions	
	Within Jupyter, open AggregateByClassification_unsolved.ipynb. Run the first two blocks of code to import your dependencies, create an instance of the MongoClient, and save the database and collection to variables.
	Write an aggregation query that counts the number of documents, grouped by "classification".
	Run the query with the aggregate method and save the results to a variable.
	Print the number of classifications in the result.
	Print the first 10 results.
	Convert the mongo result to a Pandas DataFrame and display the first 10 rows of the Pandas DataFrame.

Time’s Up! Let’s Review.

Questions?

Aggregation Pipelines

Aggregation Stages

There are many stages available to use in an aggregation pipeline, but the most commonly used stages are `$match`, `$group`, and `$sort`.

Operator	Function	Pandas Equivalent
`$match`	The equivalent of performing a `find()` query in an aggregation pipeline.	`.loc[]`
`$group`	Groups documents using aggregation accumulators.	`.groupby()`
`$sort`	Sorts the returned documents by a specified field or fields.	`.sort_values()`

Why would we create an aggregation pipeline in PyMongo rather than using `find()` and then performing the calculations in Pandas?

Why build an aggregation pipeline in MongoDB?

01

When there are hundreds or thousands of documents in a collection, using an aggregation pipeline reduces the amount of information returned.

02

e.g. If the database is stored in the cloud, when the operations are performed within the database cluster, this can save us time and significantly reduce the storage space needed on our local computers to process the data.

Instructor Demonstration

Aggregation Pipelines in PyMongo

Questions?

Activity: Build an Aggregation Pipeline

In this activity, you will practice building an aggregation pipeline in PyMongo.

Suggested Time:

10 Minutes

Activity: Build an Aggregation Pipeline

Instructions	Write a match query to find only the documents about artifacts that have a classification where "Wood" is contained in the value.
	Write an aggregation query that counts the number of documents and finds the maximum height, grouped by "classification".
	Create a dictionary that will allow the pipeline to sort by count in descending order, then sort by classification in alphabetical order.
	Create a list called pipeline that contains each stage in the pipeline process: match, group, and sort.
	Run the pipeline through the aggregate method and save the results to a variable.
Hints	Remember to use the `$regex` operator to match words.
	`$max` is the aggregation operator to find the maximum value in a field.

Activity: Build an Aggregation Pipeline

Instructions	
	Print the number of classifications in the result.
	Print the first 10 results.
	Convert the mongo result to a Pandas DataFrame.
	Display the first 10 rows of the Pandas DataFrame.

Time’s Up! Let’s Review.

Questions?


Break

Mini-Project

MongoDB Integration

31

Group Mini-Project Part 1: Import Data from an API

In this activity, you will collect additional data from the Metropolitan Museum of Art API and update the database with the new data.

Suggested Time:

15 minutes

Group Mini-Project Part 1: Import Data from an API

Connect to the Database	Import your dependencies: MongoClient from pymongo, pprint from pprint, json, and requests.
	Create an instance of MongoClient.
	Assign the met database to a variable called `db`.
	Assign the artifacts collection to a variable called `artifacts`.
Collect the Data	Run the code provided to extract the data from the Met's API about each of the artifacts in the `cave_ids` list.

Group Mini-Project Part 1: Import Data from an API

Update the Database

Choose just one item from the returned data and set it to a variable called `item_to_add`.

Pretty print `item_to_add`.

Using the `objectID` from `item_to_add`, write a query to check if the item already exists in the artifacts collection.

Using the above query, write an `if` statement that checks if the result equals `None`.

Inside your `if` statement:

- Use `insert_one` to add `item_to_add` to the artifacts collection.
- Print out the `objectID` when it is added to the collection.

Group Mini-Project Part 1: Import Data from an API

Update the Database

Combine the preceding steps to loop through the whole list of data contained in `returned_data_list` only adding to the collection when the artifact does not yet exist in the database.

Follow these steps:

- Loop through `returned_data_list`.
- Save the data from the list item to the `item_to_add` variable.
- Check if the artifact already exists in the collection.
- Inside your `if` statement:
  - Use `insert_one` to add `item_to_add` to the artifacts collection.
  - Print out the `objectID` when it is added to the collection.

Time's Up! Let's Review.

Questions?

Group Mini-Project Part 2: Aggregating the Data

In this activity, you will build a more complex query and another aggregation pipeline, then normalize your resulting DataFrame.

Suggested Time:

20 minutes

Group Mini-Project Part 2: Aggregating the Data

Instructions

Write a query that:

- Uses `find()` to find documents about artifacts that come from the "Maya" culture and returns the following fields: `accessionNumber`, `accessionYear`, `classification`, `country`, `department`, `measurements.elementMeasurements.Height`, `measurements.elementMeasurements.Width`, `measurements.elementMeasurements.Depth`, `medium`, `title`, `objectURL`.
- Uses `sort()` to sort by the artifact's height.
- Uses `limit()` to limit the number of results to 5.

Then convert the results to a Pandas DataFrame and display the DataFrame.

Build an aggregation pipeline that:

- Matches:
  - Artifacts that have a width greater than or equal to 10cm and less than 50cm.
  - Artifacts that have a height greater than or equal to 20cm and less than 60cm.
  - Artifacts that have "Sculpture" as part of their classification.
- Counts the artifacts and groups by: classification and then culture.
- Sorts by count in descending order.

Group Mini-Project Part 2: Aggregating the Data

Instructions	Pretty print the first 10 results of the aggregation pipeline.
	Write code that will extract the fields from `_id` from the aggregation pipeline to create a Pandas DataFrame that will include the following columns: classification, culture, number of artifacts. If necessary, rename and reorder the columns to match.
Note	There are multiple ways to create the Pandas DataFrame, but the easiest way is to use `pd.json_normalize()`. If you need help using this new Pandas method, check out the documentation.

Time’s Up! Let’s Review.

Questions?

Group Mini-Project Part 3: Plotting the Data

In this activity, you will use your Mongo results to integrate with Matplotlib to plot data.

Suggested Time:

20 minutes

Group Mini-Project Part 3: Plotting the Data

Instructions

You may use the starter code provided in Mongo_MiniProject_Part3.ipynb, which contains code for an aggregation pipeline and resulting normalized DataFrame, which you can practice plotting with Matplotlib. However, you are also welcome to write your own aggregation pipeline to plot the data.

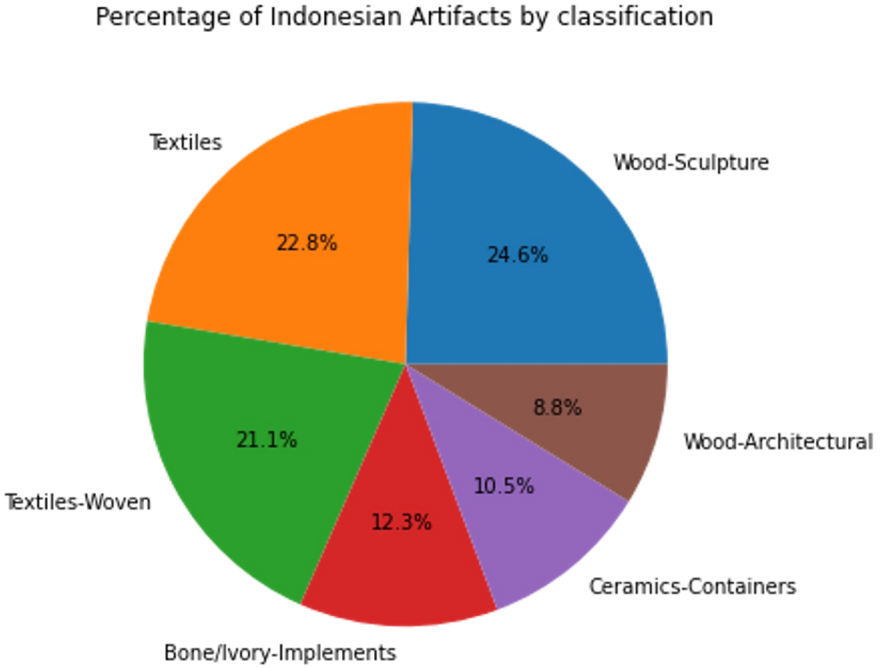
If there is enough remaining time at the end of the class, groups will be invited to present their charts to the class.

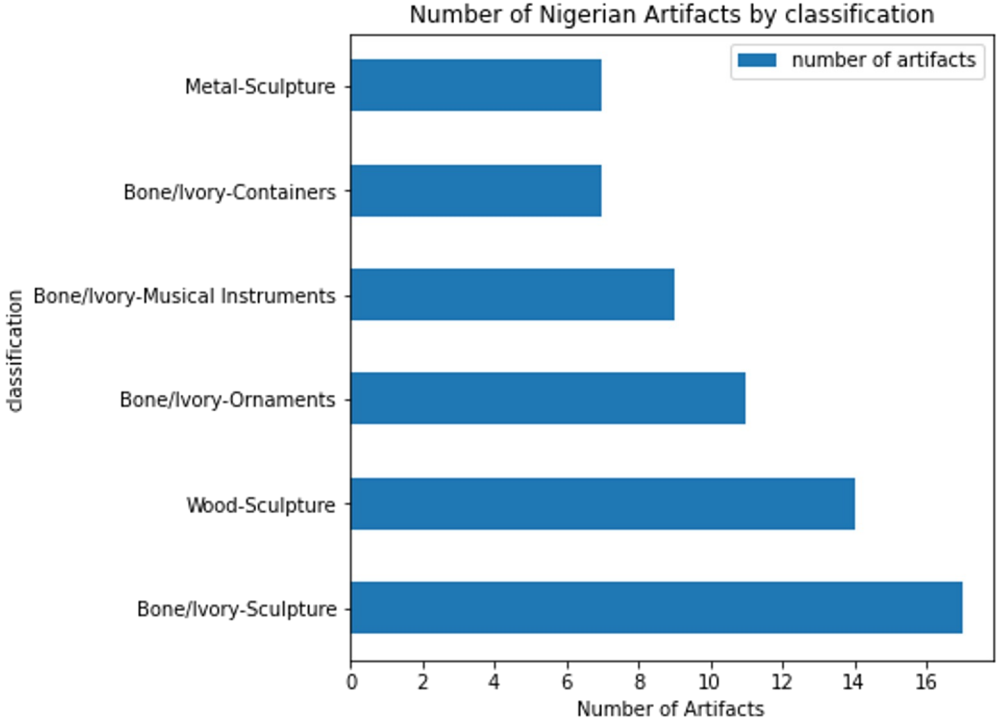
You may wish to review your Matplotlib content in order to complete the mini-project, but this is an exciting time to see how you can integrate different skills you have learned throughout the course. Feel free to also review the Pandas Plot documentation.

You are welcome to use whatever type of chart you think will best represent the results from your aggregation pipeline, or use the starter code comments as a guide.

Use `plt.savefig()` to save your chart(s).

Example Plots





Time’s Up! Let’s Review.

Questions?

Mini-Project

Mini-Presentations

48

Additional Resources

Helpful Links

Manage Mongod Processes

mongo vs mongod

PyMongo docs

Additional Resources

Helpful Terminal Commands

Find instances of Mongo `ps aux | grep mongod`

Kill process `kill -9 [pid]`

Drop Mongo Database `use <db name here>` then `db.runCommand( { dropDatabase: 1 } )`


The End.