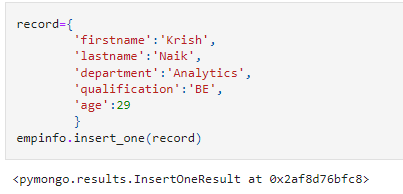
**MongoDB Practicals – Part 1**

1. **Check for the services in your system.**
2. **Check whether MongoDB server is running or not. If not start that.**
3. **Perform the following queries in spyder.**



This code connects to a MongoDB database and accesses a specific collection:

1. import pymongo: Imports the pymongo library to interact with MongoDB.
2. client = pymongo.MongoClient('mongodb://127.0.0.1:27017/'): Creates a connection to a local MongoDB server running on 127.0.0.1 (localhost) at port 27017.
3. mydb = client['Employee']: Accesses the "Employee" database on the MongoDB server.
4. empinfo = mydb.employeeinformation: Accesses the "employeeinformation" collection within the "Employee" database.



This code inserts a new record into the "employeeinformation" collection in MongoDB:

1. record = {...}: Defines a Python dictionary containing the details of an employee, including firstname, lastname, department, qualification, and age.
2. empinfo.insert\_one(record): Inserts the record dictionary into the "employeeinformation" collection. This method adds one document (record) to the collection.
3. <pymongo.results.InsertOneResult at 0x2af8d76bfc8>: This is the result returned by insert\_one(), which confirms that the record has been successfully inserted into the database. It shows the memory location of the result object.



The code defines a list of dictionaries, records, where each dictionary represents an employee's information with keys such as firstname, lastname, department, qualification, and age. Each dictionary holds data for one employee.

Here’s a breakdown of the records:

1. **First record**:
   * First Name: John
   * Last Name: Doe
   * Department: Analytics
   * Qualification: Statistics
   * Age: 35
2. **Second record**:
   * First Name: John
   * Last Name: Smith
   * Department: Analytics
   * Qualification: Masters
   * Age: 30
3. **Third record**:
   * First Name: Manish
   * Last Name: Sen
   * Department: Analytics
   * Qualification: PhD
   * Age: 34
4. **Fourth record**:
   * First Name: Ram
   * Last Name: Singh
   * Department: Analytics
   * Qualification: Master’s
   * Age: 32

This records list can now be used to insert multiple documents into the MongoDB collection (e.g., using insert\_many() method).

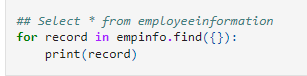


The code empinfo.find\_one() is used to retrieve a single document (record) from the employeeinformation collection in MongoDB.

Here’s what happens:

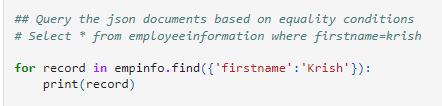
1. empinfo.find\_one(): This function retrieves the first document found in the employeeinformation collection. If no query conditions are specified, it simply returns the first document available in the collection.
2. **Returned Document**: The result is a dictionary containing the employee data, including:
   * \_id: A unique identifier automatically generated by MongoDB for each document (in this case, ObjectId('5e59ec2be0e386955e9489ba')).
   * firstname: 'Krish'
   * lastname: 'Naik'
   * department: 'Analytics'
   * qualification: 'BE'
   * age: 29

So, this code retrieves and shows the first employee record from the collection, displaying their personal details.



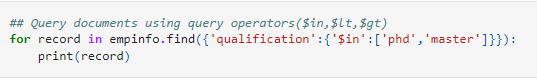
Performs a query to retrieve **all** records from the employeeinformation collection in MongoDB and prints each record.

1. empinfo.find({}):
   * The empty curly braces {} represent an **empty query filter**, which means "select all documents" from the employeeinformation collection.
   * This is similar to SELECT \* FROM employeeinformation in SQL, which retrieves all columns and rows from the table.
2. for record in empinfo.find({}):
   * The find() method returns a cursor (an iterable object) that contains all the documents in the collection that match the query (in this case, all documents).
   * The for loop iterates over each document (record) in the collection.
3. print(record):
   * Each record (employee document) is printed to the console. Each document is represented as a Python dictionary with key-value pairs.



Performs a **query with an equality condition** in MongoDB to find all documents (records) in the employeeinformation collection where the firstname field is equal to 'Krish'.

1. empinfo.find({'firstname': 'Krish'}):
   * This find() query looks for documents where the value of the firstname field is exactly 'Krish'.
   * It is equivalent to the SQL query: SELECT \* FROM employeeinformation WHERE firstname = 'Krish'.
2. for record in empinfo.find({'firstname': 'Krish'}):
   * The find() method returns a cursor (an iterable object) containing all the documents where the firstname is 'Krish'.
   * The for loop iterates over each document that matches this condition.
3. print(record):
   * Each matching document (record) is printed to the console as a Python dictionary.



performs a **query using the**$in**operator** to find documents in the employeeinformation collection where the qualification field matches either 'phd' or 'master'.

**Explanation:**

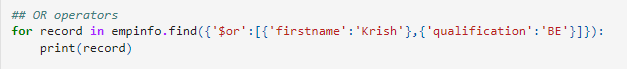
1. empinfo.find({'qualification': {'$in': ['phd', 'master']}}):
   * The query {'qualification': {'$in': ['phd', 'master']}} filters the documents by checking if the qualification field contains either 'phd' or 'master'.
   * The $in operator allows you to match a field's value against multiple possible values. In this case, it checks if the qualification is either 'phd' or 'master'.
   * This is the same as writing SQL:

sql

Copy code

SELECT \* FROM employeeinformation WHERE qualification IN ('phd', 'master');

1. for record in empinfo.find(...):
   * The find() method returns a cursor that contains all the documents that match the condition.
   * The for loop iterates through each document (record) returned by the query.
2. print(record):
   * Each matching document is printed as a Python dictionary.



Performs a **query using the**$or**operator** to find documents in the employeeinformation collection that match **either** of the two conditions:

1. The firstname is 'Krish'.
2. The qualification isBE'`.

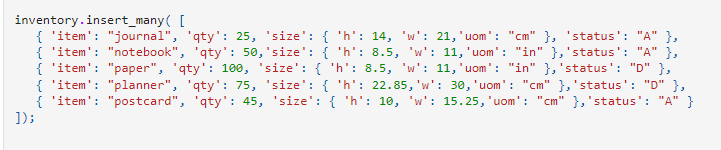
**Explanation:**

1. empinfo.find({'$or': [{'firstname': 'Krish'}, {'qualification': 'BE'}]}):
   * The $or operator allows you to specify multiple conditions, and it will match documents where **at least one** of the conditions is true.
   * In this query:
     + The first condition checks if firstname is 'Krish'.
     + The second condition checks if qualification is 'BE'.
   * This query will return all documents where either of the conditions is satisfied (or both).
2. for record in empinfo.find(...):
   * The find() method returns a cursor that contains all documents that match the $or condition.
   * The for loop iterates over each matching document (record).
3. print(record):
   * Each matching document is printed as a Python dictionary.



1. mydb.inventory:
   * This accesses the inventory collection within the mydb database. In MongoDB, a **collection** is equivalent to a table in relational databases.
   * Here, mydb is the previously defined database object (e.g., mydb = client['Employee']), and .inventory refers to the **inventory** collection within that database.
2. inventory = mydb.inventory:
   * This assigns the inventory collection to the variable inventory.
   * Now, inventory can be used to interact with this collection (e.g., to query documents, insert new items, etc.).

* The variable inventory will now hold a reference to the MongoDB collection named inventory in the mydb database, and you can use this variable to perform operations such as inserting documents, querying data, or updating records within the inventory collection.



This code inserts multiple documents (records) into the inventory collection in MongoDB. Each document represents an item in the inventory and contains information about the item's name, quantity, size, and status.

**What the code does:**

* insert\_many(): This method is used to insert multiple documents (items) into the MongoDB collection in one operation.
* Each document in the list represents an inventory item with the following fields:
  + item: The name of the item (e.g., "journal", "notebook").
  + qty: The quantity of the item in stock (e.g., 25, 50, 100).
  + size: A nested dictionary that contains:
    - h: The height of the.
    - w: The width of the item.
    - uom: The unit of measurement for the size, which can either be "cm" (centimeters) or "in" (inches).
  + status: The status of the item, which can be:
    - **"A"**: Available
    - **"D"**: Discontinued

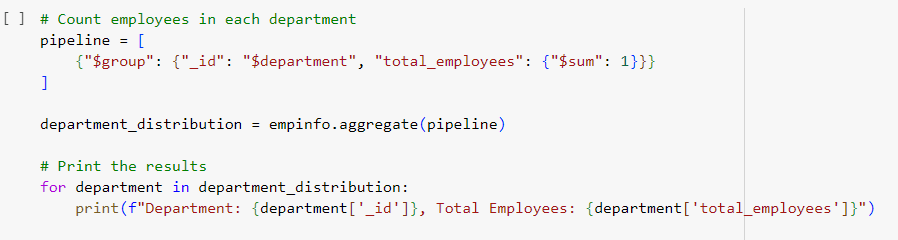


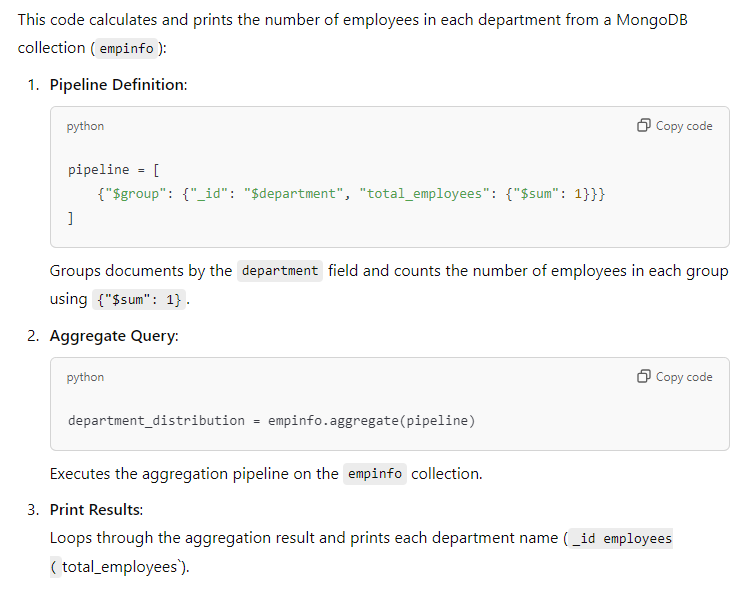
This performs a **query** on the inventory collection in MongoDB to find all documents where the size field matches the specified object:

* size: A nested dictionary containing:
  + h: Height (14 in this case)
  + w: Width (21 in this case)
  + uom: Unit of measurement ("cm" in this case)

1. inventory.find({'size': {'h': 14, 'w': 21, 'uom': "cm"}}):
   * This query looks for documents where the size field matches exactly the object {'h': 14, 'w': 21, 'uom': "cm"}.
   * In other words, it will retrieve all documents where the size field is exactly equal to this dictionary.
2. for records in inventory.find(...):
   * The find() method returns a **cursor**, which is an iterable object containing all the documents that match the query.
   * The for loop iterates through the matching documents.
3. print(records):
   * Each document that matches the query is printed to the console.

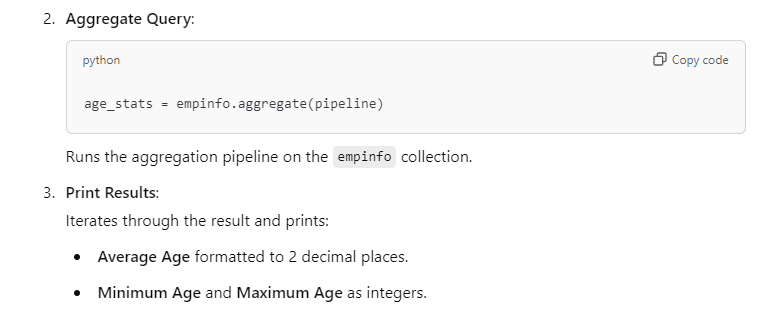
**MongoDB Practicals – Part 2**

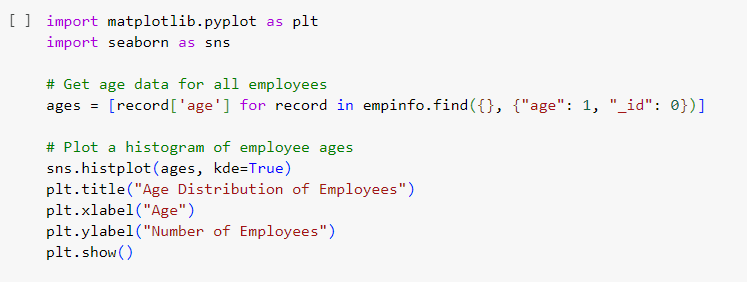






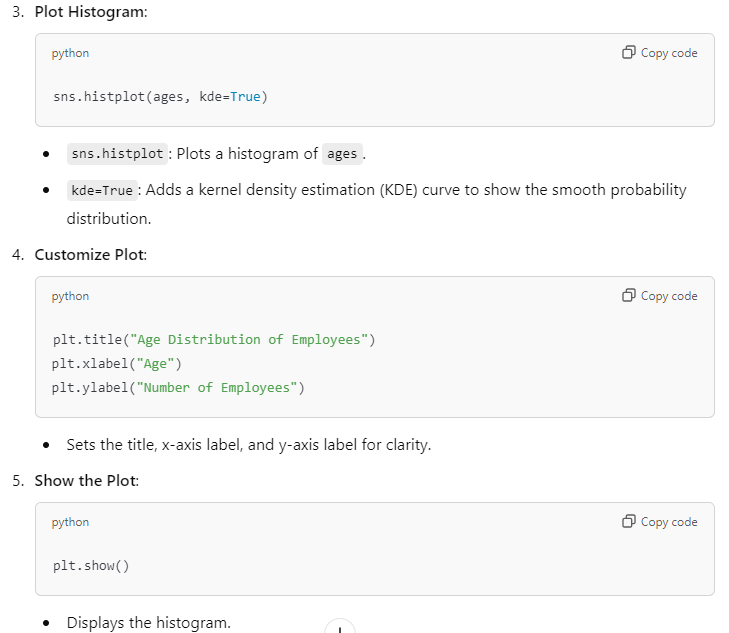




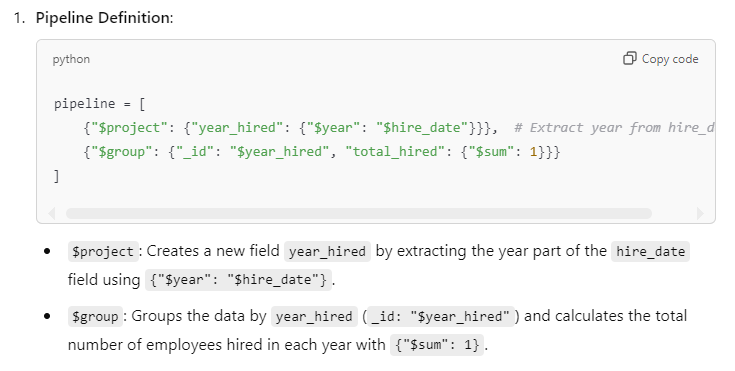


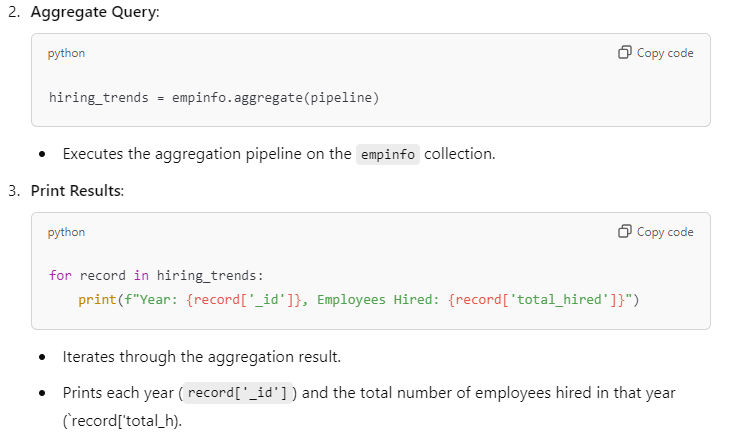
This code visualizes the distribution of employee ages in the empinfo MongoDB collection using a histogram with Seaborn.









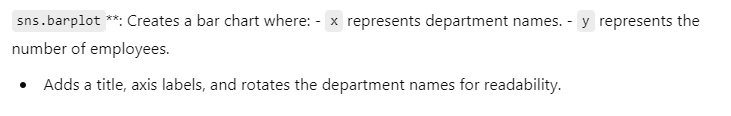




This code creates a bar chart to visualize the distribution of employees across departments in the empinfo MongoDB collection.









This code generates a line chart to visualize the hiring trends of employees over the years in the empinfo MongoDB collection.

