let’s assume that our `orders` data consists of the following structure:

```
{
    "_id": ObjectId("someuniqueid"),
    "order_id": 1,
    "customer_id": "C123",
    "product": "Laptop",
    "quantity": 1,
    "price": 1200,
    "order_date": ISODate("2023-03-01T00:00:00Z"),
    "status": "Shipped"
}
```

Here is what each field represents:

- `order_id`: A unique identifier for each order.
- `customer_id`: The ID of the customer who made the order.
- `product`: The product being ordered.
- `quantity`: The number of items in the order.
- `price`: The price per item.
- `order_date`: The date when the order was placed.
- `status`: The status of the order (e.g., "Shipped", "Pending", "Delivered").

Insert Data into MongoDB

To insert data into MongoDB, you can use the following command in the MongoDB shell or through a MongoDB client:

Example Insert in MongoDB shell

```
db.orders.insertMany([
    {
        "order_id": 1,
        "customer_id": "C123",
        "product": "Laptop",
        "quantity": 1,
        "price": 1200,
        "order_date": ISODate("2023-03-01T00:00:00Z"),
        "status": "Shipped"
    },
    {
        "order_id": 2,
        "customer_id": "C124",
        "product": "Phone",
        "quantity": 2,
        "price": 600,
        "order_date": ISODate("2023-03-02T00:00:00Z"),
        "status": "Pending"
    },
    {
        "order_id": 3,
        "customer_id": "C125",
        "product": "Tablet",
        "quantity": 3,
        "price": 400,
        "order_date": ISODate("2023-03-02T00:00:00Z"),
```

```
        "status": "Shipped"
    }
])
```

Basic Queries and Data Aggregation

Now that we have inserted some data, let's perform some basic queries and aggregation tasks.

1. Querying All Orders:

```
db.orders.find()
```

This will return all the orders in the collection.

2. Querying Orders by a Specific Customer:

To find all orders placed by a customer with `customer_id` "C123":

```
db.orders.find({ "customer_id": "C123" })
```

3. Querying Orders Based on Date Range:

If we want to find all orders placed after `2023-03-01`:

```
db.orders.find({ "order_date": { $gt: ISODate("2023-03-01T00:00:00Z") } })
```

Step 5: Advanced Data Analysis Using Aggregation Framework

The aggregation framework in MongoDB is powerful and can be used to perform complex data analysis tasks such as grouping, sorting, filtering, and calculating totals.

1. Calculating Total Sales (Sum of Prices):

Let's calculate the total sales from all the orders:

```
db.orders.aggregate([
    { $group: { _id: null, totalSales: { $sum: { $multiply: ["$quantity",
"$price"] } } } }
])
```

This query groups all orders (since `_id: null` means we're not grouping by any field) and calculates the total sales by multiplying `quantity` and `price`, then summing them.

2. Total Sales Per Product:

To calculate total sales per product, you can group the orders by the `product` field and sum the total sales for each product:

```
db.orders.aggregate([
    { $group: { _id: "$product", totalSales: { $sum: { $multiply:
["$quantity", "$price"] } } } }
])
```

This groups the orders by `product` and sums the total sales for each product.

3. Number of Orders Per Customer:

Let's find out how many orders each customer has placed:

```
db.orders.aggregate([
    { $group: { _id: "$customer_id", orderCount: { $sum: 1 } } }
])
```

This groups orders by `customer_id` and counts the number of orders for each customer using `$sum: 1`.

4. Orders by Status (Count of Pending, Shipped, Delivered Orders):

If you want to count how many orders are in each status (e.g., "Shipped", "Pending", etc.):

```
db.orders.aggregate([
    { $group: { _id: "$status", count: { $sum: 1 } } }
])
```

This groups orders by their status and counts how many orders are in each status.

5. Average Order Value:

To calculate the average order value (i.e., the average total spent per order), we can use the following aggregation:

```
db.orders.aggregate([
    { $project: { totalOrderValue: { $multiply: ["$quantity",
"$price"] } } },
    { $group: { _id: null, averageOrderValue: { $avg:
"$totalOrderValue" } } }
])
```

This first creates a new field `totalOrderValue` by multiplying `quantity` and `price`, and then calculates the average of this value.

6. Top Products by Sales:

To find the top 3 products based on total sales:

```
db.orders.aggregate([
    { $group: { _id: "$product", totalSales: { $sum: { $multiply:
["$quantity", "$price"] } } } },
    { $sort: { totalSales: -1 } },
    { $limit: 3 }
])
```

This groups the orders by `product`, sums the total sales, sorts them in descending order (`-1`), and then limits the result to the top 3 products.