🔹 1. Find the total sales (`$sum`) from the `orders` collection.

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

📌 **Explanation:** Groups all orders and calculates the sum of `totalPrice`.

🔹 2. Find the average price (`$avg`) of all products in `products`.

```
db.products.aggregate([
  { $group: { _id: null, avgPrice: { $avg: "$price" } } }])
```

📌 **Explanation:** Computes the average price of products.

🔹 3. Find the most expensive (`$max`) and cheapest (`$min`) product.

```
db.products.aggregate([
  { $group: { _id: null, maxPrice: { $max: "$price" },
minPrice: { $min: "$price" } } }])
```

📌 **Explanation:** Retrieves the highest and lowest prices.

🔹 4. Get the first (`$first`) and last (`$last`) order.

```
db.orders.aggregate([
  { $sort: { orderDate: 1 } },
  { $group: { _id: null, firstOrder: { $first: "$_id" },
lastOrder: { $last: "$_id" } } }])
```

📌 **Explanation:** Sorts by `orderDate` and selects the first and last order.

🔹 5. Calculate a 10% discount price using `$multiply` and `$subtract`.

```
db.products.aggregate([
  { $project: { name: 1, originalPrice: "$price",
discountPrice: { $multiply: ["$price", 0.9] } } }])
```

📌 **Explanation:** Multiplies `price` by `0.9` to apply a 10% discount.

🔹 6. Find orders where `totalPrice > 500` (`$gt`).

```
db.orders.aggregate([
  { $match: { totalPrice: { $gt: 500 } } }
])
```

📌 **Explanation:** Filters orders where `totalPrice` is greater than 500.

🔹 7. Find users older than 30 (`$gte`).

```
db.users.aggregate([
  { $match: { age: { $gte: 30 } } }
])
```

📌 **Explanation:** Selects users whose `age` is 30 or more.

🔹 8. Find products costing between `$50 and $500`.

```
db.products.aggregate([
  { $match: { price: { $gte: 50, $lte: 500 } } }])
```

📌 **Explanation:** Filters products within a price range.

🔹 9. Find orders that are either ‘pending’ or ‘shipped’ (`$or`).

```
db.orders.aggregate([
  { $match: { $or: [ { status: "pending" }, { status:
"shipped" } ] } }])
```

📌 **Explanation:** Matches orders where `status` is either `"pending"` or `"shipped"`.

🔹 10. Find users NOT using a Gmail email (`$not`).

```
db.users.aggregate([
  { $match: { email: { $not: /@gmail\.com$/ } } }])
```

📌 **Explanation:** Uses `$not` with regex to exclude Gmail users.

🔹 11. Concatenate first and last name (`$concat`).

```
db.users.aggregate([
  { $project: { fullName: { $concat: ["$firstName", " ",
"$lastName"] } } }
])
```

📌 **Explanation:** Combines `firstName` and `lastName`.

🔹 12. Convert product names to uppercase (`$toUpper`).

```
db.products.aggregate([
  { $project: { name: 1, upperName: { $toUpper: "$name" } } }
])
```

📌 **Explanation:** Converts `name` to uppercase.

🔹 13. Extract domain from user emails (`$substr`).

```
db.users.aggregate([
  { $project: { domain: { $substr: ["$email", { $indexOfBytes:
["$email", "@"] }, -1] } } }])
```

📌 **Explanation:** Extracts domain by finding `"@"` in `email`.

🔹 14. Find the number of orders per user (`$group & $count`).

```
db.orders.aggregate([
  { $group: { _id: "$userId", orderCount: { $count: {} } } }])
```

📌 **Explanation:** Groups orders by `userId` and counts them.

🔹 15. Find the second ordered product (`$arrayElemAt`).

```
db.orders.aggregate([
  { $project: { secondProduct: { $arrayElemAt: ["$products",
1] } } }])
```

📌 **Explanation:** Extracts the second item from `products` array.

🔹 16. Get the first 3 products (`$slice`).

```
db.products.aggregate([
  { $project: { top3Products: { $slice: ["$products", 3] } } }
])
```

📌 **Explanation:** Retrieves the first 3 elements of the `products` array.

🔹 17. Extract the year from `orderDate` (`$year`).

```
db.orders.aggregate([
  { $project: { orderYear: { $year: "$orderDate" } } }
])
```

📌 **Explanation:** Extracts the year from `orderDate`.

🔹 18. Convert `orderDate` to a custom format (`$dateToString`).

```
db.orders.aggregate([
  { $project: { formattedDate: { $dateToString: { format: "%Y-
%m-%d", date: "$orderDate" } } } }
])
```

📌 **Explanation:** Formats the date as `"YYYY-MM-DD"`.

🔹 19. Label orders as ‘High Value’ if above $500, otherwise ‘Low Value’ (`$cond`).

```
db.orders.aggregate([
  { $project: { statusLabel: { $cond: { if: { $gt:
["$totalPrice", 500] }, then: "High Value", else: "Low Value"
} } } }
])
```

📌 **Explanation:** Uses `$cond` for conditional logic.

🔹 20. Sort products by price in descending order (`$sort`).

```
db.products.aggregate([
  { $sort: { price: -1 } }
])
```

📌 **Explanation:** Sorts products from most to least expensive.