Program

Visualize data using basic plotting techniques in python

```
import matplotlib.pyplot as plt
import mumpy as np
import seaborn as sns
#1. Line plot
x = np.linespace (0, 10, 100)
y = np.sin(x)

plt.figure(figsize = (10,8))
plt.subplot(2,2,1)
plt.plot(x,y,color ='blue')
plt.title('Line Plot of y = sin(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)

#2. Bar plot
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 25]
plt.subplot(2, 2, 2)
plt.bar(categories, values, color='green')
plt.title('Bar Plot')
plt.xlabel('Category')
plt.ylabel('Values')
```

```
#3. Histogram
data = np.random.randn(1000)
plt.subplot(2,2,3)
plt.hist(data, bins = 30, edgecolor ='black', color = 'orange')
plt.title('Histogram of Normally Distributed Data')
plt.xlabel('Value')
plt.ylabel('Frequency')

#4. Scatter Plot
x_scatter = np.random.rand(50)
y-scatter = np.random.rand(50)
plt.subplot(2,2,4)
plt.scatter(x-scatter, y-scatter, color = 'red')
plt.title('Scatter Plot')
plt.xlabel('x')
plt.ylabel('y')
plt.tight_layout()
plt.show()
# Seaborn - enhanced Histogram with KDE
plt.figure(figsize = (6,4))
sns.histplot(data, kde = True, color = 'purple')
plt.title('Seaborn Histogram with KDE')
plt.show()
```

2.

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class wordcount {
  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
```

```
    private IntWritable result = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(wordcount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

3.

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class maxtemp{
public static class TempMapper
extends Mapper<LongWritable, Text,Text,IntWritable>
private Text year=new Text();
private IntWritable temperature=new IntWritable();
@Override
protected void map(LongWritable key,Text value,Context context)
IOException,InterruptedException{
String line=value.toString();
String[] fields=line.split(" ");
year.set(fields[0]);
temperature.set9Integer.parseInt(fields[1]));
context.write(year,temperature);
}
}
public static class TempReducer extends Reducer<Text,IntWritable,Text,IntWritable>
{
private IntWritable maxTemperature=new IntWritable();
```

```
@Override

protected void reduce(Text key,Iterable<IntWritable>values,Context context)throws
IOException,Interrupted Exception{

int maxTemp=Integer.MIN_VALUE;

for(IntWritable val:values){

maxTemp=Math.max(maxTemp,val.get());

}

maxTemperature.set(maxTemp);

Context.write(key,maxTemperature);

}

}

public static void main(String[] args)throws Exception{

Configuration conf=new Configuration();

Job job=Job.getInstance(conf,"Max Temperature"):

job.setJarByClass(MaxTemperature.class);

job.setReducerClass(TempReducer.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(IntWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job,new Path(args[0]));
```

4.

```
import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class StudentGrades {


  // Mapper class

  public static class GradeMapper extends Mapper<LongWritable, Text, Text, IntWritable> {


    private Text studentName = new Text();

    private IntWritable score = new IntWritable();


    @Override

    protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {

      String line = value.toString();

      String[] fields = line.split(" ");

      studentName.set(fields[0]); // Student name

      score.set(Integer.parseInt(fields[1])); // Score

      context.write(studentName, score); // Emit (name, score)

    }
    }


    // Reducer class
    public static class GradeReducer extends Reducer<Text, IntWritable, Text, Text> {


        private Text grade = new Text();


        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            for (IntWritable value : values) {
                int score = value.get();
                // Determine grade based on scoreAJITH            A


                if (score >= 90) {
                    grade.set("A");
                } else if (score >= 80) {
                    grade.set("B");
                } else if (score >= 70) {
                    grade.set("C");
                } else if (score >= 60) {
                    grade.set("D");
                } else {
                    grade.set("F");
                }
                context.write(key, grade); // Emit (student, grade)
            }
        }
    }


    // Main driver method
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Student Grades");
        job.setJarByClass(StudentGrades.class);

        // Set Mapper and Reducer classes
        job.setMapperClass(GradeMapper.class);
        job.setReducerClass(GradeReducer.class);

        // Set output key and value types for the Mapper
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(IntWritable.class);

        // Set output key and value types for the Reducer
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Input and Output paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

5

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class EvenOddCount{
  public static class EvenOddMapper extends Mapper<Object, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1); // Correct type
    private Text evenOdd = new Text();

    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
      int number=Integer.parseInt(value.toString());
      if(number%2==0){
      evenOdd.set("Even");
      }else{
      evenOdd.set("odd");
      }
      context.write(evenOdd, one);
    }
  }

  public static class EvenOddReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();
```

```
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get(); // Use 'val' instead of 'value'
            }
            result.set(sum); // Fixed typo 'resultr'
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Even odd count");
        job.setJarByClass(EvenOddCount.class); // Updated to match class name
        job.setMapperClass(EvenOddMapper.class);
        job.setCombinerClass(EvenOddReducer.class);
        job.setReducerClass(EvenOddReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

**Experiment 2 : Implement word count program using Map Reduce.**

**Steps to run WordCount Program on Hadoop:**

1. Make sure Hadoop and Java are installed properly

```
hadoop version
```

```
javac -version
```

2. Create a directory on the Desktop named Lab and inside it create two folders; one called “Input” and the other called “tutorial_classes”.

[You can do this step using GUI normally or through terminal commands]

```
cd Desktop
mkdir Lab
mkdir Lab/Input
mkdir Lab/BDA
```

3. Add the file attached with this document “WordCount.java” in the directory Lab

4. Add the file attached with this document “input.txt” in the directory Lab/Input.

5. Type the following command to export the hadoop classpath into bash.

```
export HADOOP_CLASSPATH=$(hadoop classpath)
```

Make sure it is now exported.

```
echo $HADOOP_CLASSPATH
```

7. Go to **localhost:9870** from the browser, Open “Utilities → Browse File System” and you should see the directories and files we placed in the file system.

8. Then, back to local machine where we will compile the WordCount.java file. Assuming we are currently in the Desktop directory.

9. Start the HDFS System using the command.

```
start-dfs.sh
```

10. Start the YARN using the command

```
start-yarn.sh
```

11. Type the following command. You should see an output similar to the one in the following figure.

```
jps
```

12. Make sure these nodes are listed: (ResourceManager, NameNode, NodeManager, SecondaryNameNode, Jps and DataNode).

13. It is time to create these directories on HDFS rather than locally. Type the following commands.

```
hadoop fs -mkdir /WordCountProgram
```

```
hadoop fs -mkdir /WordCountProgram/Input
```

```
hadoop fs -put Lab/Input/input.txt /WordCountProgram/Input
```

14. Go to localhost:9870 from the browser. You should expect the following

```
cd Lab
```

`javac -classpath $HADOOP_CLASSPATH -d BDA WordCount.java`Put the output files in one jar file (There is a dot at the end)

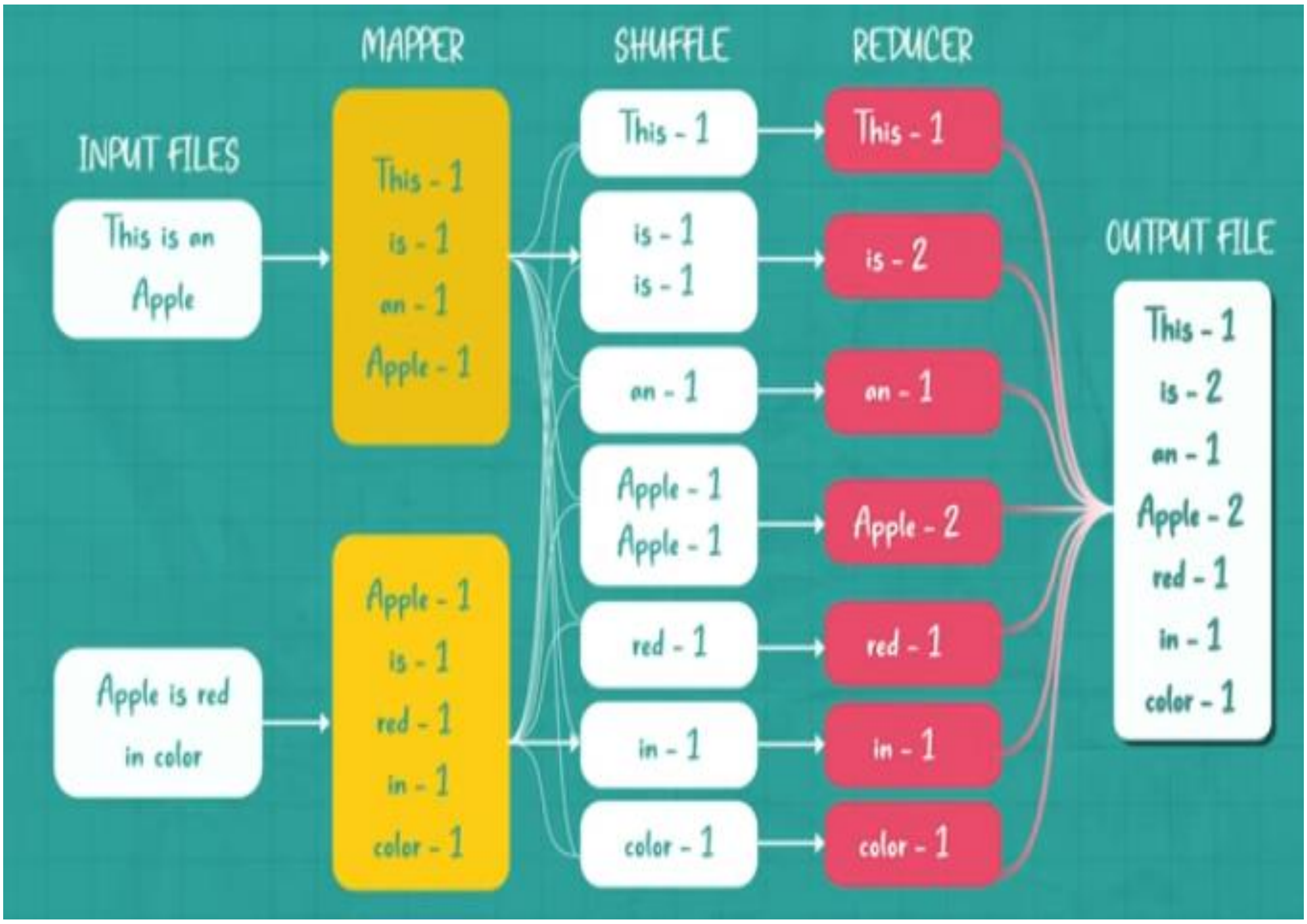
```
jar -cvf WordCount.jar -C BDA .
```

15. Now, we run the jar file on Hadoop.

```
hadoop jar WordCount.jar WordCount /WordCountProgram/Input
/WordCountProgram/Output
```

16. Output the result:

```
hadoop dfs -cat /WordCountProgram/Output/*
```


MAPPER
SHUFFLE
REDUCER
INPUT FILES
This is an Apple
Apple is red in color
This - 1
is - 1
an - 1
Apple - 1
Apple - 1
is - 1
red - 1
in - 1
color - 1
This - 1
is - 1
is - 1
an - 1
Apple - 1
Apple - 1
red - 1
in - 1
color - 1
This - 1
is - 2
an - 1
Apple - 2
red - 1
in - 1
color - 1
OUTPUT FILE
This - 1
is - 2
an - 1
Apple - 2
red - 1
in - 1
color - 1

Program 6

**Implement NoSQL Database Operations: CRUD operations, Arrays using MongoDB & Cassandra.**

Implementing CRUD operations
MongoDB supports four primary CRUD (Create, Read, Update, Delete) operations for interacting with data:

1. Create: To create a document in MongoDB, we insert it into a collection using theáinsertOne()áoráinsertMany()ámethod. Documents can contain any valid JSON data, allowing for flexible schema design.
2. Read: To read data from MongoDB, we query witháfind()ámethod. The queries filter documents based on the specified criteria and return the results.
3. Update: To update the documents, eitheráupdateOne()áoráupdateMany()ácan be used. This method facilitates flexible data manipulation.
4. Delete: To delete documents from a collection, we use theádeleteOne()áorádeleteMany()ámethod. Deletions are performed based on specified criteria, such as matching a particular field value.

NoSQL CRUD operations (Create, Read, Update, Delete) and array handling in both MongoDB and Cassandra.

```
mongosh

test> use mydb
switched to db mydb

mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    96.00 KiB


mydb> db.students.insertOne({name: "Akarsh", age: 30, cgpa: 9.0})

mydb> show dbs
admin    40.00 KiB
config  108.00 KiB
local    96.00 KiB
mydb    72.00 KiB

mydb> db.students.insertMany(
[
{name: "Ajith", age:30, cgpa:8.0},
{name: "Anush", age:30, cgpa:8.5}
]
)
```

```
mydb> db.students.find();


mydb> db.students.insertOne(
{ name: "Ananya",
age: 30,
cgpa: 9.0,
fullTime:false,
registerDate: new Date(),
gradutionDate: null,
courses: ["Cloud","BigData","Cyber"],
address: {houseNo: 123,
city: "Mlr",
zip: 575007}deleteOne({name:'Anan
}
)


mydb> db.students.find();



db.students.deleteOne({name:'Akarsh'});


mydb> db.students.find().sort( { name:1 })

mydb> db.students.find().sort( { name:-1 })

mydb> db.students.find().sort( { cgpa:1 })

mydb> db.students.find().sort( { cgpa:-1 })


mydb> show collections
students


db.student.insertOne(
{name: "Akarsh", age: 30, cgpa: 9.0})

mydb> show collections
students
student
```

## 1. Create (Insert)

MongoDB has commands to insert either a single document or multiple documents at once:

**Insert a Single Document**

```
db.collectionName.insertOne({
  name: "John Doe",
  age: 30,
  interests: ["coding", "reading"]
});
```

**Insert Multiple Documents**

```
db.collectionName.insertMany([
  { name: "Jane Doe", age: 25, interests: ["music", "traveling"] },
  { name: "Jim Beam", age: 35, interests: ["gaming", "hiking"] }
]);
```

## 2. Read (Query)

MongoDB allows you to query for specific data, apply filters, and use projections to limit which fields are returned.

**Retrieve All Documents**

```
db.collectionName.find({});
```

**Retrieve Specific Documents with a Filter**

```
db.collectionName.find({ age: { $gt: 30 } }); // Documents where age > 30
```

**Apply Projection (Limit Fields Returned)**

```
db.collectionName.find({}, { name: 1, interests: 1 }); // Only include 'name' and 'interests'
```

## 3. Update

Updating documents in MongoDB allows you to modify existing data, add fields, or work with array elements. You can update one document at a time or multiple documents that match a filter.

**Update a Single Document**

```
db.collectionName.updateOne(
  { name: "John Doe" },                    // Filter
  { $set: { age: 31 }, $push: { interests: "sports" } } // Update with new values
);
```

**Update Multiple Documents**

```
db.collectionName.updateMany(
{ age: { $lt: 30 } },
  { $set: { status: "young" } } ); // Set a new field or update existing fields
```

## 4. Delete

MongoDB supports deleting either a single document or multiple documents that meet a specific filter condition.

**Delete a Single Document**

```
db.collectionName.deleteOne({ name: "Jim Beam" });
```

**Delete Multiple Documents**

```
db.collectionName.deleteMany({ age: { $lt: 25 } });
```

## Working with Arrays in MongoDB

Arrays in MongoDB are treated as first-class citizens, and MongoDB provides various operators for managing array elements.

**Adding Elements to Arrays**

**Use the $push operator to add an element to an array field.**

```
db.collectionName.updateOne(
  { name: "John Doe" },
  { $push: { interests: "movies" } }
);
```

**You can also use $each to add multiple items to an array:**

```
db.collectionName.updateOne(
  { name: "John Doe" },
  { $push: { interests: { $each: ["swimming", "traveling"] } } }
);
```

**Removing Elements from Arrays**

**Use the $pull operator to remove an element from an array:**

```
db.collectionName.updateOne(
  { name: "John Doe" },
  { $pull: { interests: "reading" } }
);
```

**Checking for Elements in an Array**

To find documents where an array contains a specific element, use the array field directly in the query.

```
db.collectionName.find({ interests: "coding" });
```

**To check if an array contains all specified values, use $all:**

```
db.collectionName.find({ interests: { $all: ["coding", "sports"] } });
```

**Updating Specific Array Elements**

Use the positional $ operator to update the first matching array element based on a condition.

```
db.collectionName.updateOne(
  { name: "John Doe", "interests": "coding" },
  { $set: { "interests.$": "programming" } }
);
```

**Using $addToSet to Avoid Duplicates in an Array**

To add an element to an array only if it does not already exist, use $addToSet.

```
db.collectionName.updateOne(
  { name: "John Doe" },
  { $addToSet: { interests: "coding" } } // Will not add "coding" if it already exists
);
```

**Experiment 7: Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB**

| **Function** | **Method** | **Purpose** |
|---|---|---|
| Count | countDocuments() | Counts matching documents. |
| Sort | sort() | Sorts documents by fields. |
| Limit | limit() | Limits the number of documents in results. |
| Skip | skip() | Skips a specified number of documents. |
| Aggregate | aggregate() | Performs complex queries using pipelines. |

These methods provide powerful tools for querying and manipulating data in MongoDB. Depending on your requirements, you can use them individually or in combination for more complex operations.

**Insert the Collection into MongoDB: users**

```
db.users.insertMany([
  { "user_id": 1, "name": "John", "age": 30, "interests": ["coding", "sports"] },
  { "user_id": 2, "name": "Alice", "age": 25, "interests": ["reading", "music"] },
  { "user_id": 3, "name": "Bob", "age": 35, "interests": ["traveling", "sports"] },
  { "user_id": 4, "name": "Eve", "age": 28, "interests": ["cooking", "reading"] },
  { "user_id": 5, "name": "Charlie", "age": 40, "interests": ["music", "gardening"] }
]
```

**1. Count**

**Query:**

Count the number of users older than 30:

```
db.users.countDocuments({ age: { $gt: 30 } })
```

**2. Sort**

**Query:**

Sort users by age in descending order:

```
db.users.find().sort({ age: -1 })
```

**3. Limit**

**Query:**

Fetch the first 2 users:

```
db.users.find().limit(2)
```

**4. Skip**

**Query:**

Skip the first 2 users and fetch the next 2:

```
db.users.find().skip(2).limit(2)
```

**5. Aggregate**

**Query 1: Group by Age (Count Users per Age Group)**

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

Query 2: Find the Oldest User, the document with the highest age is returned.

```
db.users.aggregate([
  { $sort: { age: -1 } },
  { $limit: 1 }
])
```

Query 3: Average Age of Users

```
db.users.aggregate([
  { $group: { _id: null, averageAge: { $avg: "$age" } } }
])
```

**Combining Functions**

**Query: Fetch the second page of 2 users sorted by age in descending order**

```
db.users.aggregate([
  { $sort: { age: -1 } },
  { $skip: 2 },
  { $limit: 2 } ])
```

**Questions**

**1. Count**

Count the number of users who live in Chicago and are interested in "sports."

## 2. Sort

Sort users by `salary` in descending order, and then by `age` in ascending order (in case of ties).

**3. Limit and Skip**

Fetch the third and fourth highest-paid users:

**4. Aggregation**

**Query 1: Average Salary by City**
Find the average salary of users grouped by city:

**Query 2: Users Interested in "Sports" with Total Salary**
Find users who are interested in "sports" and calculate their total salary:

**Query 3: List Users with Selected Fields**
Return only the name, age, and salary fields of users, sorted by age in ascending order:

**5. Complex Filter: Multiple Conditions**

Find users who are:

- Above 30 years old
- Living in "Chicago" or "New York"
- Interested in "sports"

## 6. Add a New Field to Users

- Add a new field called `status` that labels users as either "High Earner" (salary >= 75000) or "Low Earner":

```
db.users.insertMany([
  { "user_id": 1, "name": "John", "age": 30, "city": "New York", "interests": ["coding", "sports"], "salary": 70000 },
  { "user_id": 2, "name": "Alice", "age": 25, "city": "Los Angeles", "interests": ["reading", "music"], "salary": 50000 },
  { "user_id": 3, "name": "Bob", "age": 35, "city": "Chicago", "interests": ["traveling", "sports"], "salary": 90000 },
  { "user_id": 4, "name": "Eve", "age": 28, "city": "San Francisco", "interests": ["cooking", "reading"], "salary": 60000 },
  { "user_id": 5, "name": "Charlie", "age": 40, "city": "New York", "interests": ["music", "gardening"], "salary": 100000 },
  { "user_id": 6, "name": "Sophia", "age": 30, "city": "Chicago", "interests": ["sports", "traveling"], "salary": 75000 }
])
```

**1. Count**

Count the number of users who live in Chicago and are interested in "sports."

```
db.users.countDocuments({ city: "Chicago", interests: "sports" })
```

**2. Sort**

Sort users by `salary` in descending order, and then by `age` in ascending order (in case of ties).

```
db.users.find().sort({ salary: -1, age: 1 })
```

**3. Limit and Skip**

Fetch the third and fourth highest-paid users:

```
db.users.find().sort({ salary: -1 }).skip(2).limit(2)
```

**4. Aggregation**

**Query 1: Average Salary by City**
Find the average salary of users grouped by city:

```
db.users.aggregate([
  { $group: { _id: "$city", avgSalary: { $avg: "$salary" } } },
  { $sort: { avgSalary: -1 } }
])
```

**Query 2: Users Interested in "Sports" with Total Salary**
Find users who are interested in "sports" and calculate their total salary:

```
db.users.aggregate([
 { $match: { interests: "sports" } },
 { $group: { _id: null, totalSalary: { $sum: "$salary" } } }
])
```

**Query 3: List Users with Selected Fields**
Return only the name, age, and salary fields of users, sorted by age in ascending order:

```
db.users.aggregate([
 { $project: { _id: 0, name: 1, age: 1, salary: 1 } },
 { $sort: { age: 1 } }
])
```

**5. Complex Filter: Multiple Conditions**

Find users who are:

- Above 30 years old
- Living in "Chicago" or "New York"
- Interested in "sports"

```
db.users.find({
 age: { $gt: 30 },
 city: { $in: ["Chicago", "New York"] },
 interests: "sports"
})
```

**6. Add a New Field to Users**

- Add a new field called `status` that labels users as either "High Earner" (salary >= 75000) or "Low Earner":

```
db.users.aggregate([
 {
  $addFields: {
   status: { $cond: { if: { $gte: ["$salary", 75000] }, then: "High Earner", else: "Low
Earner" } }
  }
 }
])
```