Experiment 1: Basic Hadoop Operations Using HDFS

Important Commands:

Create a directory in HDFS:
bash
Copy code
```
hdfs dfs -mkdir /user/cloudera/mydirectory
```

1.

Upload a file to HDFS:
bash
Copy code
```
hdfs dfs -put /path/to/localfile.txt /user/cloudera/mydirectory
```

2.

List files in an HDFS directory:
bash
Copy code
```
hdfs dfs -ls /user/cloudera/mydirectory
```

3.

Read a file from HDFS:
bash
Copy code
```
hdfs dfs -cat /user/cloudera/mydirectory/localfile.txt
```

4.

Delete a file from HDFS:
bash
Copy code
```
hdfs dfs -rm /user/cloudera/mydirectory/localfile.txt
```

5.

Experiment 2: MySQL Setup and Basic Queries

MySQL Commands:

Create a database:
sql
Copy code
```
CREATE DATABASE company;
```

1.

Use a database:
sql
Copy code
```
USE company;
```

2.

Create a table:
sql
Copy code
```
CREATE TABLE employees (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    salary DECIMAL(10, 2)
);
```

3.

Insert data into the table:
sql
Copy code
```
INSERT INTO employees (id, name, salary) VALUES (1, 'John Doe',
50000);
```

4.

Query data from the table:
sql
Copy code
```
SELECT * FROM employees;
```

5.

Experiment 3: Hive Queries

Important Hive Queries:

Create a database:
sql
Copy code
```
CREATE DATABASE testing;
```

1.

Use a database:
sql
Copy code
```
USE testing;
```

2.

Create a table:
sql
Copy code
```
CREATE TABLE posts (
    user STRING,
    post STRING,
    words BIGINT
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ',';
```

3.

Load data into a table:
sql
Copy code
```
LOAD DATA LOCAL INPATH '/home/cloudera/Desktop/DB/posts.txt' INTO
TABLE posts;
```

4.

Query the data:
sql
Copy code
```
SELECT * FROM posts;
```

5.

Experiment 4: Apache Pig

Pig Commands:

Load data:
pig
Copy code
```
posts = LOAD '/user/cloudera/input/datafile.txt' USING PigStorage(',')
AS (user:chararray, post:chararray, words:int);
```

1.

Describe the data:
pig
Copy code
```
DESCRIBE posts;
```

2.

Filter data:
pig
Copy code
```
filtered_posts = FILTER posts BY words > 10;
```

3.

Group data:
pig
Copy code
```
grouped_posts = GROUP posts BY user;
```

4.

Store the results:
pig
Copy code
```
STORE grouped_posts INTO '/user/cloudera/output/grouped_posts' USING
PigStorage(',');
```

5.

Experiment 5: MapReduce Word Count Program Using Hadoop Streaming

Python Mapper (`mapper.py`):

python
Copy code
```python
import sys
for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print(f"{word}\t1")
```

Python Reducer (`reducer.py`):

python
Copy code
```python
import sys
from collections import defaultdict

current_word = None
current_count = 0

for line in sys.stdin:
    word, count = line.split('\t')
    count = int(count)
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print(f"{current_word}\t{current_count}")
        current_word = word
        current_count = count

if current_word == word:
    print(f"{current_word}\t{current_count}")
```

Hadoop Streaming Command:

bash
Copy code
```bash
hadoop jar /usr/lib/hadoop-mapreduce/hadoop-streaming.jar \
-input /user/cloudera/input/wordcount.txt \
-output /user/cloudera/output/wordcount_output \
```

```
-mapper mapper.py \
-reducer reducer.py \
-file mapper.py \
-file reducer.py
```

Experiment 6: MongoDB Setup and NoSQL Commands

MongoDB Commands:

Show all databases:
bash
Copy code
```
show dbs;
```

1.

Create/use a database:
bash
Copy code
```
use movies;
```

2.

Insert a document:
bash
Copy code
```
db.movies.insertOne({
    title: "Inception",
    director: "Christopher Nolan",
    year: 2010
});
```

3.

Query a document:
bash
Copy code
```
db.movies.findOne({title: "Inception"});
```

4.

Experiment 7: Flajolet-Martin Algorithm

Python Code:

```
def trailing_zero(x):
    if x == 0:
        return 1
    count = 0
    while x & 1 == 0:
        count += 1
        x >>=1
    return count

def flajolet_martin(dataset, k):
    max_zeros = 0
    for _ in range(k):
        for element in dataset:
            hash_val = hash(element)
            max_zeros = max(max_zeros, trailing_zero(hash_val))


    return 2 ** max_zeros

num_elements = int(input("Enter the number of elements in the dataset:
"))

dataset = []

for i in range(num_elements):
    element = int(input(f"Enter element {i + 1}: "))
    dataset.append(element)

k = 10

dist_num = flajolet_martin(dataset, k)
print("Estimated number of distinct elements: ", dist_num)
```

Experiment 8: Bloom Filter

Python Code:

python
Copy code

```
import math


class BloomFilter:
    def __init__(self, items_count,bit_array_size):
        self.size = max(bit_array_size, 1)

        self.hash_count =
max(1,self.get_hash_count(items_count,self.size))
        self.bit_array = [False] * self.size

    def add(self,item):
        for i in range(self.hash_count):
            digest = (hash(item) + i) % self.size
            self.bit_array[digest] = True

    def check(self,item):
        for i in range(self.hash_count):
            digest = (hash(item) + i) % self.size
            if not self.bit_array[digest]:
                return False
        return True

    @classmethod
    def get_hash_count(cls,n,m):
        if n == 0 or m == 0:
            return 0
        k = (m/n) / math.log(2)
        return int(k)

if __name__ == "__main__":
    n = int(input("Enter the expected of items to add: "))
    bit_array_size = int(input("Enter the size of the bit array: "))
```

```
    bloomf = BloomFilter(n,bit_array_size)
    print("Size of bit array: {}".format(bloomf.size))
    print("Number of hash functions: {}".format(bloomf.hash_count))

    print("Enter numbers to add to the Bloom filter (type 'done to
finish):")


    while True:
        input_value = input()
        if input_value.lower() == 'done':
            break

        try:
            number = int(input_value)
            bloomf.add(number)
            print(f"Added {number} to the Bloom filter.")
        except ValueError:
            print("Please enter a valid number.")

    print("Enter numbers to check in the Bloom filter (type 'done' to
finish)")

    while True:
        input_value = input()
        if input_value.lower() == 'done':
            break
        try:
            number = int(input_value)
            if bloomf.check(number):
                print(f"'{number}' may be present (possible
positive).")
            else:
                print(f"'{number}' is definitely not present.")
        except ValueError:
            print("Please enter a valid number.")
```

Experiment 9: Data Mining Using Weka

Weka Steps:

1. **Open Weka** and load your dataset (ARFF or CSV).
2. **Preprocess**: Clean and filter data.
3. **Classify**: Use **J48** or other algorithms for classification.
4. **Cluster**: Run **K-Means** on the dataset.
5. **Associate**: Use **Apriori** for association rules.

Experiment 10: Data Visualization Using R

R Code for Data Visualization:

```
# Install required packages
install.packages("ggplot2")

# Load the library
library(ggplot2)

# Generate synthetic healthcare data
healthcare_data <- data.frame(
  PatientID = 1:100,
  Age = sample(18:85, 100, replace = TRUE),
  Gender = sample(c("Male", "Female"), 100, replace = TRUE)
)

# Plot Age Distribution
ggplot(healthcare_data, aes(x = Age)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black") +
  labs(title = "Age Distribution of Patients", x = "Age", y = "Count")
```