# BIG DATA ANALYTICS Lab manual

## 2 Develop a MapReduce program to implement Matrix Multiplication

to create a mapper file

```
nano mapper.py
```

Pyhton Code-

```
#!/usr/bin/env python3
import sys

# Define matrix dimensions (should be adapted to actual input)
A_ROWS = 2  # Number of rows in Matrix A
B_COLS = 2  # Number of columns in Matrix B

for line in sys.stdin:
    line = line.strip()
    matrix, row, col, value = line.split()
    row, col, value = int(row), int(col), int(value)

    if matrix == "A":
        # Emit for every column in B
        for k in range(B_COLS):
            print(f"{row} {k}\tA {col} {value}")

    elif matrix == "B":
        # Emit for every row in A
        for i in range(A_ROWS):
            print(f"{i} {col}\tB {row} {value}")
```

to create a reducer file

```
nano reducer.py
```

Python Code-

```
import sys
from collections import defaultdict

# Dictionary to store intermediate values
intermediate = defaultdict(list)

for line in sys.stdin:
    line = line.strip()
    key, value = line.split("\t")
    i, j = map(int, key.split())
    parts = value.split()

    intermediate[(i, j)].append((parts[0], int(parts[1]), int(parts[2])))

# Compute final matrix multiplication
for (i, j), values in intermediate.items():
    a_values = {k: v for m, k, v in values if m == "A"}
    b_values = {k: v for m, k, v in values if m == "B"}
    result = sum(a_values[k] * b_values[k] for k in a_values if k in
b_values)

    if result:
        print(f"{i} {j} {result}")
```

to create input file

```
nano matrix.txt
```

```
A 0 0 3
A 0 1 2
A 1 0 1
A 1 1 4
B 0 0 2
B 1 0 5
B 0 1 3
B 1 1 1
```

Command to run the program

```
cat matrix.txt | python mapper.py | sort | python reducer.py
```

output

```
0 0 16
0 1 11
1 0 22
1 1 7
```

**3.Develop a mapreduce program that mines weather data and displays appropriate messages including the weather condition of the day**

To create a file

```
gedit weather.py
```

Pyhton Code-

```
from mrjob.job import MRJob

class WeatherAnalysis(MRJob):
    def mapper(self, _, line):
        try:
            date, temp, humidity, precipitation = line.split(',')
            temp = float(temp)
            humidity = float(humidity)
            precipitation = float(precipitation)
            yield date, (temp, humidity, precipitation)
        except ValueError:
            pass  # Skip lines with errors

    def reducer(self, date, values):
        total_temp, total_humidity, total_precipitation = 0, 0, 0
        count = 0

        for temp, humidity, precipitation in values:
            total_temp += temp
            total_humidity += humidity
            total_precipitation += precipitation
            count += 1

        avg_temp = total_temp / count
        avg_humidity = total_humidity / count
        avg_precipitation = total_precipitation / count

        # Determine weather condition
        if avg_temp > 30:
            condition = "Hot Day"
        elif avg_temp < 10:
            condition = "Cold Day"
        elif avg_precipitation > 5:
            condition = "Rainy Day"
        elif avg_humidity > 80:
            condition = "Humid Day"
        else:
            condition = "Pleasant Day"

        yield date, f"{condition} - Avg Temp: {avg_temp:.1f}Â°C,
Humidity: {avg_humidity:.1f}%, Precipitation: {avg_precipitation:.1f}mm"

if __name__ == "__main__":
    WeatherAnalysis.run()
```

to create input file

```
nano weather_data.csv

    YYYY-MM-DD,temperature,humidity,precipitation
2025-03-07,32,60,1.2
2025-03-07,28,65,0.8
2025-03-07,35,55,0.5
2025-03-08,15,85,7.0
```

Command to execute the program

```
python weather.py weather_data.csv
```

output

```
"2025-03-07"     "Hot Day - Avg Temp: 31.7\u00b0C, Humidity: 60.0%,
Precipitation: 0.8mm"
"2025-03-08"     "Rainy Day - Avg Temp: 15.0\u00b0C, Humidity: 85.0%,
Precipitation: 7.0mm"
```

**4.Develop a mapreduce program to find the tags associated with each movie by analyzing movie lens data.**

To create a file

```
gedit movie_tag_analysis.py
```

Python Code-

```
from mrjob.job import MRJob

class MovieTagAnalysis(MRJob):
    def mapper(self, _, line):
        try:
            movie_id, tag = line.split(',')
            movie_id = movie_id.strip()
            tag = tag.strip()
            yield movie_id, tag
        except ValueError:
            pass  # Skip lines with errors

    def reducer(self, movie_id, tags):
        yield movie_id, list(tags)

if __name__ == "__main__":
    MovieTagAnalysis.run()
```

To Create Input file

```
nano movie_tags.csv
```

```
1,Action
1,Thriller
2,Comedy
2,Romance
3,Sci-Fi
3,Adventure
1,Drama
```

Command to run the program

```
python movie_tag_analysis.py movie_tags.csv
```

Output

```
"1"   ["Action","Thriller","Drama"]
"2"   ["Comedy","Romance"]
"3"   ["Sci-Fi","Adventure"]
```

**How to install apache pig**

Step 1: Download Apache Pig

```
wget https://downloads.apache.org/pig/latest/pig-0.17.0.tar.gz

tar -xvzf pig-0.17.0.tar.gz

sudo mv pig-0.17.0 /usr/local/pig
```

Step 2: Set Environment Variables

```
nano ~/.bashrc
```

Add the following lines at the end:

```
export PIG_HOME=/usr/local/pig
export PATH=$PIG_HOME/bin:$PATH
export PIG_CLASSPATH=$HADOOP_HOME/conf

export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
```

Step 3: Apply Changes

```
source ~/.bashrc
```

Step 4: Verify the Installation

```
pig -version
```

You should see output like:

```
Apache Pig version 0.17.0
```

**6.Develop Pig Latin scripts to sort, group, join, project, and filter the data.**

Create a Pig script file

```
nano script.pig

-- Load employee data
EMPLOYEES = LOAD 'employees.csv' USING PigStorage(',')
            AS (id:INT, name:CHARARRAY, age:INT, department:CHARARRAY,
salary:DOUBLE);

-- Load department data
DEPARTMENTS = LOAD 'departments.csv' USING PigStorage(',')
              AS (dept_id:CHARARRAY, dept_name:CHARARRAY);
```

```
-- Filter employees with salary > 50000
FILTERED_EMPLOYEES = FILTER EMPLOYEES BY salary > 50000;

-- Group by department
GROUPED_EMPLOYEES = GROUP FILTERED_EMPLOYEES BY department;

-- Join with department data
JOINED_DATA = JOIN FILTERED_EMPLOYEES BY department, DEPARTMENTS BY
dept_id;

-- Select required columns
PROJECTED_DATA = FOREACH JOINED_DATA GENERATE id, name, age, dept_name,
salary;

-- Sort by salary descending
SORTED_DATA = ORDER PROJECTED_DATA BY salary DESC;

-- Store output
STORE SORTED_DATA INTO 'output' USING PigStorage(',');
```

Prepare Input Data

To Create employees.csv

```
nano employees.csv
```

```
1,Alice,30,IT,80000
2,Bob,35,HR,75000
3,Charlie,28,Finance,60000
4,David,40,IT,55000
5,Eve,45,HR,45000
6,Frank,50,Marketing,70000
7,Grace,29,Finance,48000
8,Hank,33,IT,52000
```

To Create departments.csv

```
IT,Information Technology
HR,Human Resources
Finance,Finance Department
Marketing,Marketing Team
```

Command to Run Pig Script

```
pig -x local script.pig
```

```
ls output/
part-r-00000  _SUCCESS
```

```
cat output/part-r-00000
```

Output

```
1,Alice,30,Information Technology,80000.0
```

```
2,Bob,35,Human Resources,75000.0
6,Frank,50,Marketing Team,70000.0
3,Charlie,28,Finance Department,60000.0
4,David,40,Information Technology,55000.0
8,Hank,33,Information Technology,52000.0
```

Implement a word count program in hadoop and spark

word count program in Hadoop
To create a file

```
gedit worcount.py
```

Python Code

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"\b\w+\b")

class WordCountMR(MRJob):

    def mapper(self, _, line):
        """Emit each word with a count of 1"""
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def reducer(self, word, counts):
        """Sum up counts for each word"""
        yield word, sum(counts)

if __name__ == "__main__":
    WordCountMR.run()
```

To create input file

```
nano input.txt
```

```
hai how are you
i m fine
hai how about you
```

To run the program

```
python wordcount.py input.txt
```

output

```
"about" 1
"are" 1
"fine"      1
"hai" 2
"how" 2
"i"   1
```

```
"m"   1
"you" 2
```

**8.Implement a word count program in hadoop and spark**

word count program in Hadoop
To create a file

```
gedit worcount.py
```

Python Code

```
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"\b\w+\b")

class WordCountMR(MRJob):

    def mapper(self, _, line):
        """Emit each word with a count of 1"""
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def reducer(self, word, counts):
        """Sum up counts for each word"""
        yield word, sum(counts)

if __name__ == "__main__":
    WordCountMR.run()
```

To create input file

```
nano input.txt
```

```
hai how are you
i m fine
hai how about you
```

To run the program

```
python wordcount.py input.txt
```

output

```
"about" 1
"are" 1
"fine"      1
"hai" 2
"how" 2
"i"   1
"m"   1
"you" 2
```

word count program in spark

To create a file

```
gedit wordcountspark.py
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split, col

# Initialize Spark session
spark = SparkSession.builder.appName("WordCount").getOrCreate()

# Read input text file
text_rdd = spark.read.text("input.txt").rdd

# Perform word count using RDD transformations
word_counts = (
    text_rdd.flatMap(lambda line: line[0].split())  # Split lines into
words
    .map(lambda word: (word.lower(), 1))           # Map each word to
(word, 1)
    .reduceByKey(lambda a, b: a + b)               # Reduce by key (word)
)

# Collect and print results
for word, count in word_counts.collect():
    print(f"{word}: {count}")

# Stop Spark session
spark.stop()
```

To create input file

```
nano input.txt
```

```
hai how are you
i m fine
hai how about you
```

To run the program

```
spark-submit wordcountspark.py
```

output

```
hai: 2
how: 2
are: 1
you: 2
i: 1
m: 1
fine: 1
```

```
about: 1
```

word count program in spark

To create a file

```
gedit wordcountspark.py
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, split, col

# Initialize Spark session
spark = SparkSession.builder.appName("WordCount").getOrCreate()

# Read input text file
text_rdd = spark.read.text("input.txt").rdd

# Perform word count using RDD transformations
word_counts = (
    text_rdd.flatMap(lambda line: line[0].split())  # Split lines into
words
    .map(lambda word: (word.lower(), 1))           # Map each word to
(word, 1)
    .reduceByKey(lambda a, b: a + b)               # Reduce by key (word)
)

# Collect and print results
for word, count in word_counts.collect():
    print(f"{word}: {count}")

# Stop Spark session
spark.stop()
```

To create input file

```
nano input.txt
```

```
hai how are you
i m fine
hai how about you
```

To run the program

```
spark-submit wordcountspark.py
```

output

```
hai: 2
```

```
how: 2
are: 1
you: 2
i: 1
m: 1
fine: 1
about: 1
```

Program 5,7 and 9 to update