Exp 1:

Hadoop fs -mkdir /user/exp1

Hadoop fs -ls /user

Hadoop fs -put /home/cloudera/Desktop/data.txt /user/exp1

Hadoop fs -ls /user/exp1

Hadoop fs -get /user/exp1/data.txt /home/cloudera/Desktop/temp/

Dir /home/cloudera/Desktop/temp

Haddop fs -copyFromLocal /home/cloudera/Desktop/temp /user/exp1/dir

Hadoop fs -copyToLocal /user/exp1/dir/data.txt /home/cloudera/Desktop/temp

Haddop fs -cp /user/exp1/data.txt /user/exp/

Hadoop fs -mv /user/exp/data.txt /user/exp/dir/data.txt

Hadoop fs -rm /user/exp/dir/data.txt

Hadoop fs -cat /user/exp1/data.txt

Hadoop fs -tail /user/exp1/data.txt

Hadoop fs -du /user/exp1/data.txt

Exp 2a

Matrix Multiplication:

public class MatrixVectorMultiplication {

public static void main(String[] args) {

// Define the matrix and vector

int[][] matrix = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

int[] vector = {2, 3, 4};

// Check if matrix and vector dimensions are compatible

int matrixRows = matrix.length;

int matrixCols = matrix[0].length;

int vectorSize = vector.length;

if (matrixCols != vectorSize) {

System.out.println("Matrix and vector dimensions are not compatible for

multiplication.");

return;

}

// Perform matrix-vector multiplication

int[] result = new int[matrixRows];

for (int i = 0; i < matrixRows; i++) {

for (int j = 0; j < matrixCols; j++) {

result[i] += matrix[i][j] \* vector[j];

}

}

// Display the result

System.out.println("Result of matrix-vector multiplication:");

for (int i = 0; i < matrixRows; i++) {

System.out.println(result[i]); }}}

Javac Matrix.java -cp $(hadoop classpath)

Jar -cvf multi.jar \*.class

Hadoop jar multi.jar Matrix /home/cloudera/out

Exp 2b

Word starting with ‘h’

Rdd = sc.textFile(“file:/home/cloudera/Desktop/data.txt”)

Rdd = rdd.flatMap(lambda line:line.split())

Rdd = rdd.filter(lambda word:word.startswith(‘h’))

Rdd = rdd.map(lambda word: (word,1))

Rdd = rdd.reduceByKey(lambda x,y: x+y)

rdd.collect()

For Word Count

rdd = sc.textFile(“file:/home/cloudera/Desktop/data.txt”)

Rdd = rdd.flatMap(lambda line:line.split())

Rdd = rdd.map(lambda word: (word,1))

Rdd = rdd.reduceByKey(lambda a,b: a+b))

rdd.collect()

For Selection:  
From pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

Data = [[‘John’, 28],[‘Rachel’,24]]

Rdd = sqlContext.createDataFrame(data, [‘name’,’age’])

rdd.select(‘name’).show()

For Projection:

Data = sc.parallelize([[‘abc’,1],[‘bcd’,2],[‘cdf’,3]])

Data = Data.map(lambda x: (x[0])

Print “Projection -> %s” %(data.collect())

Union:

S = sc.parallelize([1,2,3,4,5,6]

R = sc.parallelize([5,6,7,8,0])

Uni = s.union(r)

uni.collect()

Aggregation and Grouping:

Average:

rdd.agg({“age”:”avg”}).show()

Count:

List = sc.parallelize([1,2],[1,4],[2,6],[2,3],[4,1],[5,4]])

Rdd = list.map(lambda x: (x[0], 1))

Rdd = list.reduceByKey(lambda x,y: x+y)

rdd.collect()

Max and min element:

Max\_element = list.reduceByKey(lambda x,y: max(x,y))

Min\_element = list.reduceByKey(lambda x,y: min(x,y))

Sum:

Sum = list.map(lambda x: (x[0],x[1]))

Sum = list.reduceByKey(lambda x,y: x+y)

sum.collect()

Join:

Create 2 dataframe

Ta = TableA.alias(‘ta’)

Tb = TableB.alias(‘tb’)

Inner\_join = ta.join(tb, ta.name == tb.name)

Left\_join = ta.join(tb, ta.name == tb.name, how=’left’)

Right\_join = ta.join(tb, ta.name == tb.name, how=’right’)

Intersection:

U = s+r

U = u.map(lambda x: (x,1))

U = u.reducebyKey( lambda x,y: x+y)

U = u.filter(lambda y:y[1] >1)

Exp 3:

Mysql -u root -p cloudera

Create database sales;

Use sales;

Create table sales…

Load data infile ‘/home/cloudera/Desktop/sales.csv’ into table sales fields terminated by ‘,’ lines terminated by ‘\n’;

Sqoop list-tables –connect jdbc:mysql://localhost/sales -u root -p “cloudera”

Sqoop import –connect jdbc:mysql://localhost/sales –u root -p cloudera –table = sales –target-dir = /sales/sales –incremental append –check-column month\_number –fields-terminated-by=’\t’;

Sqoop import-all-tables –connect jdbc:mysql://localhost/sales - u root -p cloudera –compression-codec=snappy –as-parquetfile –warehouse-dir = /user/hive/warehouse –hive-import

Hive

Show Tables;

Hbase shell

Version

Create ‘customer’, ‘address’, ‘order’

List

Describe ‘customer’

Put ‘customer’,’ABC’, ‘address:city’, “mumbai”

Put ‘customer’, ‘ABC’, ‘order:no”, ‘101’

Scan ‘customer’

Count ‘customer’

Get ‘customer’, ‘abc’

Delete ‘customer’, ‘abc’

Truncate ‘customer’

Disable ‘customer’

Drop ‘customer’

Exp 5:

Use net download forest fire csv

Hdfs dfs -put /home/cloudera/Desktop/forestfires.csv /user/cloudera

Sudo hive

Create external table forestfire(x int, y int, month string,....) row format delimited fields terminated by ‘,’;

Load data inpath ‘,,,’ overwrite into table forestfire;

Run sql queries;

Select \* from forestfire limit 10;

Select month, avg(ffmc) as average from forestfire group by month;.

Select \* from forestfire where x=7 and y=4 limit 10;