BELOW ARE THE COMMANDS EACH

PRACTICAL EXAM QUESTIONS

FOR HADOOP BASIC COMMANDS

Question 1:

```
# Create a directory named "data" in HDFS
!hdfs dfs -mkdir data

# Create an empty file named "sample.txt" inside the "data" directory
!hdfs dfs -touchz data/sample.txt

# List the contents of the "data" directory
!hdfs dfs -ls data

# Upload a local file named "localfile.txt" from your local filesystem to the "data" directory in HDFS
!echo "This is a local file." > localfile.txt
!hdfs dfs -put localfile.txt data/

# Retrieve the "sample.txt" file from HDFS to your local filesystem
!hdfs dfs -get data/sample.txt
```

Question 2:

```
# Create a directory named "logs" in HDFS
!hdfs dfs -mkdir logs

# Create a sample log file named "server.log" in the local filesystem
!echo "This is a server log file." > server.log

# Upload the "server.log" file from the local filesystem to the "logs" directory in HDFS
!hdfs dfs -put server.log logs/

# Display the contents of the "server.log" file stored in HDFS
!hdfs dfs -cat logs/server.log

# Rename the "server.log" file to "app_logs.txt" in HDFS
!hdfs dfs -mv logs/server.log logs/app_logs.txt

# Remove the "app_logs.txt" file from HDFS
!hdfs dfs -rm logs/app_logs.txt
```

Question 3:

```
# Create a directory named "input" in HDFS
!hdfs dfs -mkdir input

# Create sample text files "file1.txt" and "file2.txt" in the local filesystem
!echo "Content of file1.txt" > file1.txt
!echo "Content of file2.txt" > file2.txt

# Upload multiple local files (file1.txt, file2.txt) to the "input" directory in HDFS
!hdfs dfs -put file1.txt file2.txt input/

# Check the existence of the uploaded files in the "input" directory
!hdfs dfs -test -e input/file1.txt && echo "File exists"
!hdfs dfs -test -e input/file2.txt && echo "File exists"

# Download all files from the "input" directory in HDFS to a local directory named "downloads"
!hdfs dfs -get input/* downloads/

# Remove all files from the "input" directory in HDFS
!hdfs dfs -rm input/*
```

Question 4:

```
# Create a directory named "documents" in HDFS
!hdfs dfs -mkdir documents

# Create a sample document file named "report.docx" in the local filesystem
!echo "This is a report document." > report.docx

# Upload the "report.docx" file from the local filesystem to the "documents" directory in HDFS
!hdfs dfs -put report.docx documents/

# List the contents of the "documents" directory to verify the upload
!hdfs dfs -ls documents

# Move the "report.docx" file to a new directory named "archived" within the "documents" directory
!hdfs dfs -mv documents/report.docx documents/archived/

# Remove the "archived" directory from HDFS
!hdfs dfs -rmr documents/archived/
```

Question 5:

```
# Create a directory named "images" in HDFS
!hdfs dfs -mkdir images

# Uploading a sample image file "photo.jpg" to Google Colab environment (replace this with your actual image file)
# Then upload to HDFS
!echo "This is a sample image." > photo.jpg
!hdfs dfs -put photo.jpg images/

# Downloading the "photo.jpg" file from HDFS to the local filesystem
!hdfs dfs -get images/photo.jpg

# Create a duplicate copy of the "photo.jpg" file named "backup.jpg" in the "images" directory in HDFS
!hdfs dfs -cp images/photo.jpg images/backup.jpg

# Display the contents of the "images" directory to verify the presence of both files
!hdfs dfs -ls images/
```

Question 6:

```
# Create a directory named "videos" in HDFS
!hdfs dfs -mkdir videos

# Uploading a sample video file "movie.mp4" to Google Colab environment (replace this with your actual video file)
# Then upload to HDFS
!echo "This is a sample video." > movie.mp4
!hdfs dfs -put movie.mp4 videos/

# Move the "movie.mp4" file to a new directory named "archive" within the "videos" directory
!hdfs dfs -mv videos/movie.mp4 videos/archive/

# Rename the "archive" directory to "old_movies"
!hdfs dfs -mv videos/archive videos/old_movies

# Delete the "old_movies" directory from HDFS
!hdfs dfs -rmr videos/old_movies
```

Question 7:

```
# Create a directory named "temp" in HDFS
!hdfs dfs -mkdir temp

# Create sample text files "file1.txt", "file2.txt", and "file3.txt" in the local filesystem
!echo "Content of file1.txt" > file1.txt
!echo "Content of file2.txt" > file2.txt
!echo "Content of file3.txt" > file3.txt

# Upload multiple local files (file1.txt, file2.txt, file3.txt) to the "temp" directory in HDFS
!hdfs dfs -put file1.txt file2.txt file3.txt temp/

# List all files in the "temp" directory along with their details
!hdfs dfs -ls -h temp/

# Delete one of the uploaded files from the "temp" directory
!hdfs dfs -rm temp/file1.txt

# Display the remaining files in the "temp" directory
!hdfs dfs -ls -h temp/
```

Question 8:

```
# Create a directory named "analytics" in HDFS
!hdfs dfs -mkdir analytics

# Create an empty file named "data.txt" inside the "analytics" directory
!hdfs dfs -touchz analytics/data.txt

# List the contents of the "analytics" directory to verify the creation of the file
!hdfs dfs -ls analytics/

# Uploading a sample CSV file "stats.csv" to Google Colab environment (replace this with your actual CSV file)
# Then upload to HDFS
!echo "Column1,Column2,Column3" > stats.csv
!hdfs dfs -put stats.csv analytics/

# Retrieve the "stats.csv" file from HDFS to your local filesystem
!hdfs dfs -get analytics/stats.csv
```

Question 9:

```
# Create a directory named "logs" in HDFS
!hdfs dfs -mkdir logs

# Create a sample log file named "server.log" in the local filesystem
!echo "This is a server log file." > server.log

# Upload the "server.log" file from the local filesystem to the "logs" directory in HDFS
!hdfs dfs -put server.log logs/

# Display the contents of the "server.log" file stored in HDFS
!hdfs dfs -cat logs/server.log

# Move the "server.log" file to a new directory named "archive" within the "logs" directory
!hdfs dfs -mv logs/server.log logs/archive/

# Remove the "archive" directory along with its contents
!hdfs dfs -rmr logs/archive/
```

Question 10:

```
# Upload a local file named "document.docx" to the root directory ("/") in HDFS
!mkdir document.docx
!hdfs dfs -put document.docx /

# Copy the "document.docx" file from HDFS to your local filesystem
!hdfs dfs -get /document.docx

# List the contents of the root directory ("/") in HDFS to verify the presence of the file
!hdfs dfs -ls /

# Move the "document.docx" file to a new directory named "documents" in HDFS
!hdfs dfs -mv /document.docx /documents

# Delete the "documents" directory along with its contents
!hdfs dfs -rmr /documents
```

Question 11:

```
# Create a directory named "output" in HDFS
!hdfs dfs -mkdir output

# Create a sample result file named "results.txt" in the local filesystem
!echo "This is the result." > results.txt

# Upload the "results.txt" file from the local filesystem to the "output" directory in HDFS
!hdfs dfs -put results.txt output/

# Display the contents of the "results.txt" file stored in HDFS
!hdfs dfs -cat output/results.txt

# Rename the "results.txt" file to "final_results.txt" in HDFS
!hdfs dfs -mv output/results.txt output/final_results.txt

# Remove the "final_results.txt" file from HDFS
!hdfs dfs -rm output/final_results.txt
```

Question 12:

```
# Create a directory named "input_files" in HDFS
!hdfs dfs -mkdir input_files

# Create sample text files "file1.txt" and "file2.txt" in the local filesystem
!echo "Content of file1.txt" > file1.txt
!echo "Content of file2.txt" > file2.txt

# Upload multiple local files (file1.txt, file2.txt) to the "input_files" directory in HDFS
!hdfs dfs -put file1.txt file2.txt input_files/

# List all files in the "input_files" directory along with their details
!hdfs dfs -ls -h input_files/

# Copy one of the uploaded files to a new location within HDFS
!hdfs dfs -cp input_files/file1.txt input_files/copied_file.txt

# Delete all files from the "input_files" directory in HDFS
!hdfs dfs -rmr input_files/*
```

Question 13:

```
# Create a directory named "temp_files" in HDFS
!hdfs dfs -mkdir temp_files

# Create a sample temp file named "temp.txt" in the local filesystem
!echo "This is a temp file." > temp.txt

# Upload the "temp.txt" file from the local filesystem to the "temp_files" directory in HDFS
!hdfs dfs -put temp.txt temp_files/

# Display the contents of the "temp.txt" file stored in HDFS
!hdfs dfs -cat temp_files/temp.txt

# Rename the "temp.txt" file to "new_temp.txt" in HDFS
!hdfs dfs -mv temp_files/temp.txt temp_files/new_temp.txt

# Remove the "new_temp.txt" file from HDFS
!hdfs dfs -rm temp_files/new_temp.txt
```

HERE ARE PRACTICAL EXAM QUESTIONS

WITH PROGRAMS FOR HADOOP

Question 1:

IMPORT DATA FROM MYSQL INTO HDFS

AIM :

To Import Data from Mysql into HDFS.

PROCEDURE :

Step – 1 :- Installing jdk and Hadoop

Step - 2 :- Installing sqoop

Step - 3 :- Installing Mysql-Connector

Step - 4 :- Installing Mysql Server

Step - 5 :- Display All Packages Versions

Command :-

```
[ ]   !java -version
      !hadoop version
      !mysql --version
      !sqoop version
```

Output:-

```
openjdk version "11.0.22" 2024-01-16
OpenJDK Runtime Environment (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1)
OpenJDK 64-Bit Server VM (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1, mixed mode, sharing)

Hadoop 2.10.2
Subversion Unknown -r 965fd380006fa78b2315668fbc7eb432e1d8200f
Compiled by ubuntu on 2022-05-24T22:35Z
Compiled with protoc 2.5.0
From source with checksum d3ab737f7788f05d467784f0a86573fe
This command was run using /content/hadoop-2.10.2/share/hadoop/common/hadoop-common-2.10.2.jar

mysql Ver 8.0.36-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))

Sqoop 1.4.6
git commit id c0c5a81723759fa575844a0a1eae8f510fa32c25
Compiled by root on Mon Apr 27 14:38:36 CST 2015
```

Step - 6 :- Create a new directory folder ‘ afsal ’

Command :-

```
[ ]   !hdfs dfs -mkdir afsal
```

Output:-

```
Directory created
```

Step – 7 :- Create a Mysql Database and Table

Command :-

```
[ ]  import mysql.connector
     from tabulate import tabulate

     conn = mysql.connector.connect(user='root', password='root', host='localhost', port='3306')

     cursor = conn.cursor()

     cursor.execute("CREATE DATABASE IF NOT EXISTS library")

     cursor.execute("USE library")

     cursor.execute(' ' ' CREATE TABLE books (  id INT PRIMARY KEY,
                                                title VARCHAR (255),
                                                author VARCHAR (255),
                                                year_published INT) ' ' ')

     books_data = [ (1,"To Kill a Mockingbird", "Harper Lee", 1960),
                    (2,"1984", "George Orwell", 1949),
                    (3,"The Great Gatsby", "F. Scott Fitzgerald", 1925) ]

     cursor.executemany(' ' ' INSERT INTO books (id,title, author, year_published)
                                VALUES   (%s,%s, %s, %s) ' ' ', books_data)

     conn.commit()

     cursor.execute("SELECT * FROM books")

     records = cursor.fetchall()

     columns = [desc[0] for desc in cursor.description]

     print(tabulate(records, headers=columns, tablefmt="grid"))

     cursor.close()
     conn.close()
```

Output:-

```
+-------+--------------------------- +---------------------- +----------------------+
|  id   | title                      | author                |  year_published      |
+=====+=================== +================+==============+
|   1   | To Kill a Mockingbird      | Harper Lee            |         1960         |
+------ +----------------------------+-----------------------+----------------------+
|   2   | 1984                       | George Orwell         |         1949         |
+------ +--------------------------- +-----------------------+----------------------+
|   3   | The Great Gatsby           | F. Scott Fitzgerald   |         1925         |
+------ +--------------------------- +-----------------------+----------------------+
```

Step – 8 :- Import Data From Mysql to HDFS using sqoop

Command :-

```
[ ]  !sqoop import \
         --connect jdbc:mysql://localhost:3306/library \
         --username root \
         --password root \
         --table books \
         --target-dir afsal/mysql_book \
         --m 1
```

Output:-

Step - 9: - List all Files in a Folder

Command :-

```
[ ]    !hdfs dfs -ls afsal/mysql_book
```

Output:-

```
Found 2 items
-rw-r--r--   1 root root          0 2024-03-16 14:07 afsal/mysql_book/_SUCCESS
-rw-r--r--   1 root root        110 2024-03-16 14:07 afsal/mysql_book/part-m-00000
```

Step - 10: - Display the Imported Table in HDFS

Command :-

```
[ ]    !hdfs dfs -cat afsal/mysql_book/part-m-00000
```

Output:-

```
1,To Kill a Mockingbird,Harper Lee,1960
2,1984,George Orwell,1949
3,The Great Gatsby,F. Scott Fitzgerald,1925
```

RESULT :

Thus, The Data from Mysql into HDFS has been Successfully Imported.

Question 2:

EXPORTING DATA FROM HDFS TO MYSQL

AIM :

To Exporting Data from HDFS To Mysql.

PROCEDURE :

Step – 1 :- Installing jdk and Hadoop

Step - 2 :- Installing sqoop

Step - 3 :- Installing Mysql-Connector

Step - 4 :- Installing Mysql Server

Step - 5 :- Display All Packages Versions

Command :-

```
[ ]   !java -version
      !hadoop version
      !mysql --version
      !sqoop version
```

Output:-

```
openjdk version "11.0.22" 2024-01-16
OpenJDK Runtime Environment (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1)
OpenJDK 64-Bit Server VM (build 11.0.22+7-post-Ubuntu-0ubuntu222.04.1, mixed mode, sharing)

Hadoop 2.10.2
Subversion Unknown -r 965fd380006fa78b2315668fbc7eb432e1d8200f
Compiled by ubuntu on 2022-05-24T22:35Z
Compiled with protoc 2.5.0
From source with checksum d3ab737f7788f05d467784f0a86573fe
This command was run using /content/hadoop-2.10.2/share/hadoop/common/hadoop-common-2.10.2.jar

mysql  Ver 8.0.36-0ubuntu0.22.04.1 for Linux on x86_64 ((Ubuntu))

Sqoop 1.4.6
git commit id c0c5a81723759fa575844a0a1eae8f510fa32c25
Compiled by root on Mon Apr 27 14:38:36 CST 2015
```

Step - 6 :- Create a new directory folder ‘ export_table ’

Command :-

```
[ ]   !hdfs dfs -mkdir export_table
```

Output:-

Directory created

Step – 7 :- Create a Mysql Database and Table

Command :-

```
[ ]  import mysql.connector
     from tabulate import tabulate

     conn = mysql.connector.connect(user='root', password='root', host='localhost', port='3306')
     cursor = conn.cursor()
     cursor.execute("CREATE DATABASE IF NOT EXISTS Mysql_DB")
     cursor.execute("USE Mysql_DB")
         create_table = ' ' 'CREATE TABLE employee ( ID INT,
                                                     FIRST_NAME CHAR (20) NOT NULL,
                                                      LAST_NAME CHAR (20),
                                                      DEPT CHAR (20),
                                                      YEAR INT,
                                                      EMAIL CHAR (50) )' ' '
     cursor.execute(create_table)

     conn.commit()

     cursor.execute("SELECT * FROM EMPLOYEE")

     result = cursor.fetchall()

     columns = [desc[0] for desc in cursor.description]

     print(tabulate(result, headers=columns, tablefmt="grid"))

     cursor.close()
     conn.close()
```

Output:-

```
+------+--------------+-------------+---------+--------+--------------------+
|   ID | FIRST_NAME   | LAST_NAME   | DEPT    |   YEAR | EMAIL              |
+======+==============+=============+=========+========+====================+
|      |              |             | BCA(DS) |      2 |                    |
+------+--------------+-------------+---------+--------+--------------------+
```

Step - 8 :- Create a text file ‘ table.txt ’ from ‘ export_table ’ directory and add content

Command :-

```
[ ]  %%writefile export_table/table.txt

     1,Abul,A,BCA(DS),2,abul@gmail.com

     2,Afsal,A,BCA(DS),2,afsal@gmail.com

     3,Amanulla,mallick,BCA(DS),2,aman@gmail.com

     4,Anilesh,C,BCA(DS),2,anilesh@gmail.com
```

Output:-

```
Writing export_table/table.txt
```

Step - 9 :- Display the content ‘ export_table/table.txt ’

Command :-

```
[ ]  !hdfs dfs -cat export_table/table.txt
```

Output:-

1,Abul,A,BCA(DS),2,abul@gmail.com

2,Afsal,A,BCA(DS),2,afsal@gmail.com

3,Amanulla,mallick,BCA(DS),2,aman@gmail.com

4,Anilesh,C,BCA(DS),2,anilesh@gmail.com

Step - 10 :- Exporting Data From HDFS to Mysql using Sqoop

Command :-

```
[ ]  !sqoop export \
       --connect jdbc:mysql://localhost:3306/Mysql_DB \
       --username root \
       --password root \
       --table employee \
       --export-dir export_table \
       --m 1
```

Output:-

Step - 11: - Display the Exported Table in Mysql

Command :-

```
[ ]  conn = mysql.connector.connect(user='root', password='root', host='localhost',
     port='3306', database='Mysql_DB')

     cursor = conn.cursor()

     cursor.execute("SELECT * FROM EMPLOYEE")

     result = cursor.fetchall()

     columns = [desc[0] for desc in cursor.description]

     print(tabulate(result, headers=columns, tablefmt="grid"))
```

Output:-

ID	FIRST_NAME	LAST_NAME	DEPT	YEAR	EMAIL
1	Abul	A	BCA(DS)	2	abul@gmail.com
2	Afsal	A	BCA(DS)	2	afsal@gmail.com
3	Amanulla	Mallick	BCA(DS)	2	aman@gmail.com
4	Anilesh	C	BCA(DS)	2	anilesh@gmail.com

RESULT :

Thus, the data from HDFS to MySQL has been Successfully Exported.

Question 3 :

FILE MANAGEMENT TASKS IN HADOOP.

AIM :

To File Management Tasks in Hadoop.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop

Step - 2 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 3 :- Create a text file ‘ file.txt ’ from ‘ Input ’ directory and add content

Command :-

```
[ ]    %%writefile input/file.txt
       Hello Everyone !
```

Output :-

Writing input/file.txt

Step - 4 :- List all the files in ‘ input ’ directory

Command :-

```
[ ]    !hdfs dfs -ls input
```

Output :-

Found 1 items
-rw-r--r-- 1 root root 594 2024-04-6 13:51 input/file.txt

Step - 5 :- Display ‘ file.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/file.txt
```

Output :-

Hello Everyone !

Step - 6 :- Move File content from ‘ file.txt ’ to ‘ file1.txt ’

Command :-

```
[ ]    !hdfs dfs -mv input/file.txt   file1.txt
```

Output :-

File content is moved

Step - 7 :- Display ‘ file1.txt ’ File content

Command :-

```
[ ]    !cat   file1.txt
```

Output :-

Hello Everyone !

Step - 8 :- Copy a File content from ‘ file1.txt ’ to ‘ input ’

Command :-

```
[ ]    !cp   file1.txt  input/
```

Output :-

File content is copied

Step - 9 :- Display copied File content form ‘ input/file1.txt ’

Command :-

```
[ ]    !hdfs dfs -cat input/file1.txt
```

Output :-

Hello Everyone !

Step - 10 :- Remove a File ‘ file1.txt ’

Command :-

```
[ ]    !hdfs dfs -rm input/file1.txt
```

Output :-

2024-03-16 16:42:01,982 INFO Configuration.deprecation: io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
Deleted input/file1.txt

Step - 11 :- Remove a Directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -rm -r input
```

Output :-

```
2024-03-16 16:42:37,451 INFO Configuration.deprecation: io.bytes.per.checksum is deprecated.
Instead, use dfs.bytes-per-checksum
Deleted input
```

RESULT :

Thus, The File Management Tasks in Hadoop has been executed successfully

Question 4 :

WORD COUNT MAP REDUCE PROGRAM TO UNDERSTAND MAP-REDUCE PARADIGM

AIM :

To Word Count Map Reduce Program to Understand Map-Reduce Paradigm.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop

Step - 2 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 3 :- Create a new file ‘ text.txt ’ from ‘ input ’ directory and add content

Command :-

```
[ ]    %%writefile input/text.txt

       Hadoop. One of the first frameworks to address the requirements of big data analytics, Apache
       Hadoop is an open-source ecosystem that stores and processes large data sets through a
       distributed computing environment. Hadoop can scale up or down, depending on your needs,
       which makes it a highly flexible and cost-efficient framework for managing big data
```

Output :-

Writing input/text.txt

Step - 4 :- List all the files in ‘ input ’ directory

Command :-

```
[ ]    !hdfs dfs -ls input
```

Output :-

```
Found 1 items
-rw-r--r--   1 root root        594 2024-03-16 13:51 input/text.txt
```

Step - 5 :- Display ‘ text.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/text.txt
```

Output :-

Hadoop. One of the first frameworks to address the requirements of big data analytics, Apache Had oop is an open-source ecosystem that stores and processes large data sets through a distributed com puting environment. Hadoop can scale up or down, depending on your needs, which makes it a hig hly flexible and cost-efficient framework for managing big data

Step - 6 :- Mapreduce the ‘ text.txt ’ file content

Command :-

```
[ ]  !yarn jar /content/hadoop-3.3.6/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar wordcount input output
```

Output :-

Step - 7 :- Display count the words

Command :-

```
[ ]  !cat /content/output/part-r-00000
```

Output :-

Apache	1
Hadoop	2
Hadoop.	1
One	1
a	2
address	1
an	1
analytics,	1
and	2
big	2
can	1
computing	1
cost-efficient	1
data	3
depending	1
distributed	1
down,	1
ecosystem	1
environment.	1
first	1
flexible	1
for	1
framework	1
frameworks	1
highly	1
is	1
it	1
large	1
makes	1
managing	1
needs,	1
of	2
on	1
open-source	1
or	1
processes	1
requirements	1

scale	1
sets	1
stores	1
that	1
the	2
through	1
to	1
up	1
which	1
your	1

RESULT :

Thus, The Word Count Map Reduce Program Successfully tokenized input records into word count occurrences.

Question 5 :

FIND GRADE USING HADOOP MAP-REDUCE

AIM :

To Find Grade Using Hadoop Map-reduce.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop

Step - 2 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]  %%writefile mapper.py
     #!/usr/bin/env python

     import sys

     for line in sys.stdin:
        line = line.strip()

        columns = line.split(',')

        if len(columns) == 3:
           student_id, name, marks = columns
           total_marks = sum(map(float, [marks]))

           if total_marks >= 90:
              grade = "A+"
           elif total_marks >= 80:
              grade = "A"
           elif total_marks >= 70:
              grade = "B+"
           elif total_marks >= 60:
              grade = "B"
           elif total_marks >= 50:
              grade = "C"
           else:
              grade = "U"

           print(f"{student_id},{name},{total_marks},{grade}")
```

Output :-

Writing mapper.py

Step - 3 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
[ ]   %%writefile reducer.py
      #!/usr/bin/env python

      import sys

      column_names = ["S-ID", "Name", "Total Marks", "Grade"]

      print('{:10} {:<11} {:<16} {:<15}'.format(*column_names))
      print()

      for line in sys.stdin:
          line = line.strip()
          student_id, name, total_marks, grade = line.split(',')
          print(f"{student_id:<10} {name:<15} {total_marks:<15} {grade}")
```

Output :-

```
Writing reducer.py
```

Step - 4 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]   !hdfs dfs -mkdir input
```

Output:-

```
Directory created
```

Step - 5 :- Create a text file ‘ marksheet.txt ’ from ‘ input ’ directory and add content

Command :-

```
[ ]   %%writefile input/marksheet.txt

      01,John,85
      02,Emma,92
      03,Michael,78
      04,Sophia,60
      05,William,88
```

Output :-

```
Writing input/marksheet.txt
```

Step - 6 :- Run Finding Grade through hadoop mapreduce

Command :-

```
[ ]   !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
        -files mapper.py,reducer.py \
        -mapper mapper.py \
        -reducer reducer.py \
        -input input \
        -output output
```

Output :-

Step - 7 :- Display output for Map Reduced Finding Grade

Command :-

```
[ ]   !cat /content/output/part-00000
```

Output :-

```
S-ID        Name          Total Marks       Grade

01          John              85.0            A
02          Emma              92.0            A+
03          Michael           78.0            B+
04          Sophia            60.0            B
05          William           88.0            A
29          Gabriel           68.0            B
30          Victoria          98.0            A+
```

RESULT :

Thus, The Finding Grade with Hadoop Map-Reduce is successfully executed

Question 6 :

ANALYSE WEATHER DATA SET AND PRINT WHETHER THE DAY IS SHINNY OR COOL

AIM :

To Analyse Weather Data Set and Print Whether the Day is Shinny or Cool Day.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop

Step - 2 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]  %%writefile mapper.py
     #!/usr/bin/env python

     import sys

     for line in sys.stdin:
        date, temperature = line.strip().split(',')

        if int(temperature) >= 25:
           weather = 'sunny'
        else:
           weather = 'cool'

        print(f"{date}\t{weather}")
```

Output :-

Writing mapper.py

Step - 3 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
%%writefile reducer.py
#!/usr/bin/env python

import sys

current_date = None
current_weather = None

print("{:<15} {:<15}".format("Date", "Weather"))
print()

for line in sys.stdin:
   date, weather = line.strip().split('\t', 1)
```

```
        if current_date and current_date != date:
            print(f"{current_date}\t{current_weather}")

        current_date = date
        current_weather = weather
```

Output :-

Writing reducer.py

Step - 4 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 5 :- Create a text file ‘ weather.txt ’ from ‘ input ’ directory and add content

Command :-

```
[ ]    %%writefile input/weather.txt

       2024-04-01,20
       2024-04-02,28
       2024-04-03,22
       2024-04-04,30
       2024-04-05,18
       2024-04-06,25
       2024-04-07,20
       2024-04-08,27
       2024-04-09,24
       2024-04-10,29
```

Output :-

Writing input/ weather.txt

Step - 6 :- Run to Analyse Whether the Day is Shinny or Cool Day through hadoop mapreduce

Command :-

```
[ ]    !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
         -files mapper.py,reducer.py \
         -mapper mapper.py \
         -reducer reducer.py \
         -input input \
         -output output
```

Output :-

Step - 7 :- Display output for Map Reduced Analyzed Whether the Day is Shinny or Cool Day

Command :-

```
[ ]    !cat /content/output/part-00000
```

Output :-

```
Date            Weather

2024-04-01      cool
2024-04-02      sunny
2024-04-03      cool
2024-04-04      sunny
2024-04-05      cool
2024-04-06      sunny
2024-04-07      cool
2024-04-08      sunny
2024-04-09      cool
2024-04-10      sunny
```

RESULT :

Thus, The Analyse Weather Data Set and Print Whether the Day is Shinny or Cool Day with Hadoop Map-Reduce is successfully executed.

Question 7 :

IMPLEMENTING MATRIX MULTIPLICATION WITH HADOOP MAP-REDUCE

AIM :

To Implementing Matrix Multiplication with Hadoop Map-Reduce.

PROCEDURE :

Step – 1 :- Installing jdk and Hadoop

Step – 2 :- Download the text file 'matrix.txt' from GitHub

Step – 3 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]   %%writefile mapper.py
      #!/usr/bin/env python
      import sys

      m = 2
      p = 3

      for line in sys.stdin:
          entry = line.strip().split(",")
          if len(entry) == 4:
              row = int(entry[1])
              col = int(entry[2])
              value = float(entry[3])
              if entry[0] == "A":
                  for k in range(p):
                      print('{0:d},{1:d}\tA,{2:d},{3:f}'.format(row, k, col, value))
              elif entry[0] == "B":
                  for k in range(m):
                      print('{0:d},{1:d}\tB,{2:d},{3:f}'.format(k, col, row, value))
```

Output :-

Writing mapper.

Step – 4 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
[ ]  %%writefile reducer.py
#!/usr/bin/env python

import sys

n = 5

current_key = None
current_res = 0.0
value_dict = {}

for line in sys.stdin:
    line = line.strip()

    key, value = line.split('\t', 1)

    try:
        row, col = map(int, key.split(','))
        value = value.split(',')
        replicate_key, element_value = int(value[1]), float(value[2])
        key = (row, col)
    except:
        continue

    if key == current_key:
        if replicate_key not in value_dict:
            value_dict[replicate_key] = [element_value]
        else:
            value_dict[replicate_key].append(element_value)

    else:
        if current_key:
            for j in range(n):
                if j in value_dict and len(value_dict[j]) == 2:
                    current_res += value_dict[j][0] * value_dict[j][1]
            print('({0:d},{1:d}),{2:f}'.format(row, col, current_res))

        current_key = key
        value_dict = {}

        value_dict[replicate_key] = [element_value]
        current_res = 0.0

if current_key:
    for j in range(n):
        if j in value_dict and len(value_dict[j]) == 2:
            current_res += value_dict[j][0] * value_dict[j][1]
    print('({0:d},{1:d}),{2:f}'.format(row, col, current_res))
```

Output :-

```
Writing reducer.py
```

Step - 5 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 6 :- Move text File from ‘ matrix.txt’ to ‘ input/ ’

Command :-

```
[ ]    !hdfs dfs -mv  matrix.txt  input/
```

Output :-

text File is moved

Step - 7 :- Display ‘ matrix.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/matrix.txt
```

Output :-

A,0,0,0.0
A,0,1,1.0
A,0,2,2.0
A,0,3,3.0
A,0,4,4.0
A,1,0,5.0
A,1,1,6.0
A,1,2,7.0
A,1,3,8.0
A,1,4,9.0
B,0,0,0.0
B,0,1,1.0
B,0,2,2.0
B,1,0,3.0
B,1,1,4.0
B,1,2,5.0
B,2,0,6.0
B,2,1,7.0
B,2,2,8.0
B,3,0,9.0
B,3,1,10.0
B,3,2,11.0
B,4,0,12.0
B,4,1,13.0
B,4,2,14.0

Step – 8 :- Run Matrix mulitplication through hadoop mapreduce

Command :-

```
[ ]   !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
        -files mapper.py,reducer.py \
        -mapper mapper.py \
        -reducer reducer.py \
        -input input \
        -output output
```

Output :-

Step – 9 :- Display output for Map Reduced Matrix multiplication

Command :-

```
[ ]   !cat /content/output/part-00000
```

Output :-

```
(0,1),90.000000
(0,2),100.000000
(1,0),110.000000
(1,1),240.000000
(1,2),275.000000
(1,2),310.000000
```

RESULT :

Thus, The Implementing Matrix Multiplication with Hadoop Map-Reduce is successfully executed

Question 8 :

MAXIMUM TEMPERATURE IN EACH YEAR USING HADOOP MAP-REDUCE

AIM :

To find maximum Temperature in Each Year Using Hadoop Map-reduce.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop
Step – 2 :- Download the text file ‘ temperature.txt ' from GitHub
Step - 3 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]   %%writefile mapper.py
      #!/usr/bin/env python

      import sys

      for line in sys.stdin:
          parts = line.strip().split(",")
          if len(parts) == 2:
              print(f"{parts[0][:4]}\t{parts[1]}")
```

Output :-

Writing mapper.py

Step - 4 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
[ ]   %%writefile reducer.py
      #!/usr/bin/env python

      import sys

      current_year = None
      max_temperature = -float('inf')

      print("{:<10} {:<15}".format("Year", "Max Temperature"))
      print()

      for line in sys.stdin:
          line = line.strip()
          year, temperature = line.split("\t")
          temperature = float(temperature)
```

```
            if current_year == year:
                max_temperature = max(max_temperature, temperature)
            else:
                if current_year is not None:
                    print("{:<10} {:<15}".format(current_year, max_temperature))
                current_year = year
                max_temperature = temperature

        if current_year is not None:
            print("{:<10} {:<15}".format(current_year, max_temperature))
```

Output :-

Writing reducer.py

Step - 5 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 6 :- Move text File from ‘ temperature.txt ’ to ‘ input/ ’

Command :-

```
[ ]    !hdfs dfs -mv  temperature.txt  input/
```

Output :-

text File is moved

Step - 7 :- Display ‘ temperature.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/temperature.txt
```

Output :-

2010-01-01,20
2010-01-02,22
2010-01-03,18
2010-01-04,26
2010-01-05,21
2011-01-01,24
2011-01-02,26
2011-01-03,31
2011-01-04,28
2011-01-05,25
2012-01-01,22
2012-01-02,29

2012-01-03,20
2012-01-04,26
2012-01-05,23
2013-01-01,26
2013-01-02,28
2013-01-03,24
2013-01-04,33
2013-01-05,27
2014-01-01,28
2014-01-02,35
2014-01-03,26
2014-01-04,32
2014-01-05,29
2015-01-01,30
2015-01-02,37
2015-01-03,28
2015-01-04,34
2015-01-05,31
2016-01-01,32
2016-01-02,34
2016-01-03,30
2016-01-04,39
2016-01-05,33
2017-01-01,41
2017-01-02,36
2017-01-03,32
2017-01-04,38
2017-01-05,35
2018-01-01,36
2018-01-02,38
2018-01-03,34
2018-01-04,43
2018-01-05,37
2019-01-01,38
2019-01-02,40
2019-01-03,45
2019-01-04,42
2019-01-05,39
2020-01-01,40
2020-01-02,47
2020-01-03,38
2020-01-04,44
2020-01-05,41
2021-01-01,42
2021-01-02,44
2021-01-03,40
2021-01-04,46
2021-01-05,49
2022-01-01,44
2022-01-02,46
2022-01-03,51
2022-01-04,48
2022-01-05,45
2023-01-01,46
2023-01-02,48
2023-01-03,53
2023-01-04,50
2023-01-05,47

Step – 8 :- Run to Max Temperature in Each Year through hadoop mapreduce

Command :-

```
[ ]   !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
        -files mapper.py,reducer.py \
        -mapper mapper.py \
        -reducer reducer.py \
        -input input \
        -output output
```

Output :-

Step - 9 :- Display output for Map Reduced Max Temperature in Each Year

Command :-

```
[ ]   !cat /content/output/part-00000
```

Output :-

```
Year        Max Temperature

2010        26.0
2011        31.0
2012        29.0
2013        33.0
2014        35.0
2015        37.0
2016        39.0
2017        41.0
2018        43.0
2019        45.0
2020        47.0
2021        49.0
2022        51.0
2023        53.0
```

RESULT :

Thus, The Maximum Temperature in Each Year with Hadoop Map-Reduce is successfully executed.

Question 9 :

MAXIMUM ELECTRICAL CONSUMPTION IN EACH YEAR USING HADOOP MAP-REDUCE

AIM :

To find maximum electrical Consumption in Each Year Using Hadoop Map-reduce.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop
Step - 2 :- Download the text file ‘ consumption.txt ' from GitHub
Step - 3 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]  %%writefile mapper.py
     #!/usr/bin/env python

     import sys

     for line in sys.stdin:
        parts = line.strip().split(",")
        if len(parts) == 2:
           print(f"{parts[0][:4]}\t{parts[1]}")
```

Output :-

Writing mapper.py

Step - 4 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
[ ]  %%writefile reducer.py
     #!/usr/bin/env python
     import sys

     current_year = None
     max_consumption = -float('inf')

     print("{:<10} {:<15}".format("Year", "Consumption"))
     print()

     for line in sys.stdin:
        line = line.strip()
        year, consumption = line.split("\t")
        consumption = float(consumption)
```

```
        if current_year == year:
            max_consumption = max(max_consumption, consumption)
        else:
            if current_year is not None:
                print("{:<10} {:<15}".format(current_year, max_consumption))
            current_year = year
            max_consumption = consumption

    if current_year is not None:
        print("{:<10} {:<15}".format(current_year, max_consumption))
```

Output :-

Writing reducer.py

Step - 5 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 6 :- Move text File from ‘ consumption.txt’ to ‘ input/ ’

Command :-

```
[ ]    !hdfs dfs -mv  consumption.txt  input/
```

Output :-

text File is moved

Step - 7 :- Display ‘ consumption.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/consumption.txt
```

Output :-

```
2010-01-01,150
2010-01-02,160
2010-01-03,170
2010-01-04,240
2010-01-05,210
2011-01-01,245
2011-01-02,220
2011-01-03,211
2011-01-04,238
2011-01-05,225
2012-01-01,232
```

2012-01-02,249
2012-01-03,250
2012-01-04,246
2012-01-05,223
2013-01-01,236
2013-01-02,228
2013-01-03,255
2013-01-04,233
2013-01-05,247
2014-01-01,238
2014-01-02,235
2014-01-03,260
2014-01-04,232
2014-01-05,259
2015-01-01,230
2015-01-02,237
2015-01-03,258
2015-01-04,334
2015-01-05,265
2016-01-01,232
2016-01-02,244
2016-01-03,260
2016-01-04,239
2016-01-05,270
2017-01-01,241
2017-01-02,266
2017-01-03,272
2017-01-04,280
2017-01-05,275
2018-01-01,266
2018-01-02,278
2018-01-03,264
2018-01-04,280
2018-01-05,277
2019-01-01,285
2019-01-02,270
2019-01-03,275
2019-01-04,282
2019-01-05,279
2020-01-01,260
2020-01-02,287
2020-01-03,278
2020-01-04,290
2020-01-05,281
2021-01-01,282
2021-01-02,294
2021-01-03,287
2021-01-04,295
2021-01-05,289
2022-01-01,297
2022-01-02,288
2022-01-03,300
2022-01-04,298
2022-01-05,295
2023-01-01,302
2023-01-02,298
2023-01-03,350
2023-01-04,340
2023-01-05,347

Step - 8 :- Run to Max Electrical Consumption in Each Year through hadoop mapreduce

Command :-

```
[ ]  !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
       -files mapper.py,reducer.py \
       -mapper mapper.py \
       -reducer reducer.py \
       -input input \
       -output output
```

Output :-

Step - 9 :- Display output for Map Reduced Max Electrical Consumption in Each Year

Command :-

```
[ ]  !cat /content/output/part-00000
```

Output :-

```
Year        Max consumption

2010        240.0
2011        245.0
2012        250.0
2013        255.0
2014        260.0
2015        265.0
2016        270.0
2017        275.0
2018        280.0
2019        285.0
2020        290.0
2021        295.0
2022        300.0
2023        305.0
```

RESULT :

Thus, The Maximum Electrical Consumption in Each Year with Hadoop Map-Reduce is successfully executed.

Question 10 :

FIND TAGS ASSOCIATED WITH EACH MOVIE BY ANALYSING MOVIE LENS DATA USING MAP-REDUCE

AIM :

To Find Tags Associated With Each Movie By Analysing Movie Lens Data Using Hadoop Map-Reduce

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop
Step - 2 :- Download the text file ‘ movie_lens_data.txt ' from GitHub
Step - 3 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]  %%writefile mapper.py
     #!/usr/bin/env python

     import sys

     for line in sys.stdin:
        fields = line.strip().split("::")
        if len(fields) == 5:
           movie_id, movie_name, tags = fields[1], fields[2], fields[3].split(",")
           for tag in tags:
              print('%s\t%s\t%s' % (movie_id, movie_name, tag))
```

Output :-

Writing mapper.py

Step - 4 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
[ ]  %%writefile reducer.py
     #!/usr/bin/env python

     import sys

     rows = {}

     print('{:<10} {:<60} {:<20}'.format("MovieID", "Moviename", "Tags"))
     print()

     for line in sys.stdin:
        movie_id, movie_name, tag = line.strip().split('\t')
```

```
        if movie_id in rows:
            rows[movie_id][1] += f", {tag}"
        else:
            rows[movie_id] = [movie_name, tag]

    for movie_id, (movie_name, tags) in rows.items():
        print('{:<10} {:<60} {:<20}'.format(movie_id, movie_name, tags))
```

Output :-

Writing reducer.py

Step - 5 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 6 :- Move text File from ‘ movie_lens_data.txt ’ to ‘ input/ ’

Command :-

```
[ ]    !hdfs dfs -mv  movie_lens_data.txt  input/
```

Output :-

text File is moved

Step - 7 :- Display ‘ movie_lens_data.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/ movie_lens_data.txt
```

Output :-

```
UserID::MovieID::Moviename::Tag::Timestamp
01::01::Spider-Man: No Way Home (2021)::action::1648847400
01::01::Spider-Man: No Way Home (2021)::adventure::1648847400
02::02::Dune (2021)::sci-fi::1648847400
02::02::Dune (2021)::fantasy::1648847400
03::03::The Matrix Resurrections (2021)::action::1648847400
03::03::The Matrix Resurrections (2021)::sci-fi::1648847400
04::04::Black Widow (2021)::action::1648847400
04::04::Black Widow (2021)::adventure::1648847400
05::05::Shang-Chi and the Legend of the Ten Rings (2021)::action::1648847400
05::05::Shang-Chi and the Legend of the Ten Rings (2021)::fantasy::1648847400
06::06::No Time to Die (2021)::action::1648847400
06::06::No Time to Die (2021)::adventure::1648847400
07::07::Eternals (2021)::action::1648847400
07::07::Eternals (2021)::fantasy::1648847400
08::08::Free Guy (2021)::comedy::1648847400
08::08::Free Guy (2021)::adventure::1648847400
```

09::09::Jungle Cruise (2021)::adventure::1648847400
09::09::Jungle Cruise (2021)::fantasy::1648847400
10::10::Venom: Let There Be Carnage (2021)::action::1648847400
10::10::Venom: Let There Be Carnage (2021)::sci-fi::1648847400
11::11::Inception (2010)::sci-fi::1648847400
11::11::Inception (2010)::thriller::1648847400
12::12::The Dark Knight (2008)::action::1648847400
12::12::The Dark Knight (2008)::crime::1648847400
13::13::Interstellar (2014)::sci-fi::1648847400
13::13::Interstellar (2014)::adventure::1648847400
14::14::Fight Club (1999)::drama::1648847400
14::14::Fight Club (1999)::psychological::1648847400
15::15::The Shawshank Redemption (1994)::drama::1648847400
15::15::The Shawshank Redemption (1994)::inspirational::1648847400
16::16::Pulp Fiction (1994)::crime::1648847400
16::16::Pulp Fiction (1994)::black comedy::1648847400
17::17::Forrest Gump (1994)::drama::1648847400
17::17::Forrest Gump (1994)::romance::1648847400
18::18::The Godfather (1972)::crime::1648847400
18::18::The Godfather (1972)::mafia::1648847400
19::19::The Lord of the Rings: The Fellowship of the Ring (2001)::fantasy::1648847400
19::19::The Lord of the Rings: The Fellowship of the Ring (2001)::adventure::1648847400
20::20::The Matrix (1999)::action::1648847400
20::20::The Matrix (1999)::cyberpunk::1648847400

Step - 8 :- Run to Find Tags Associated with Each Movie by Analysing Movie Lens Data through hadoop mapreduce

Command :-

```
[ ]  !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
       -files mapper.py,reducer.py \
       -mapper mapper.py \
       -reducer reducer.py \
       -input input \
       -output output
```

Output :-

Step - 9 :- Display output for Map Reduced Find Tags Associated with Each Movie

Command :-

```
[ ]  !cat /content/output/part-00000
```

Output :-

MovieID	Moviename	Tags
01	Spider-Man: No Way Home (2021)	adventure, action
02	Dune (2021)	sci-fi, fantasy
03	The Matrix Resurrections (2021)	sci-fi, action
04	Black Widow (2021)	action, adventure
05	Shang-Chi and the Legend of the Ten Rings (2021)	fantasy, action
06	No Time to Die (2021)	action, adventure
07	Eternals (2021)	fantasy, action
08	Free Guy (2021)	comedy, adventure

09	Jungle Cruise (2021)	fantasy, adventure
10	Venom: Let There Be Carnage (2021)	sci-fi, action
11	Inception (2010)	thriller, sci-fi
12	The Dark Knight (2008)	crime, action
13	Interstellar (2014)	adventure, sci-fi
14	Fight Club (1999)	psychological, drama
15	The Shawshank Redemption (1994)	inspirational, drama
16	Pulp Fiction (1994)	black comedy, crime
17	Forrest Gump (1994)	romance, drama
18	The Godfather (1972)	mafia, crime
19	The Lord of the Rings: The Fellowship of the Ring (2001)	adventure, fantasy
20	The Matrix (1999)	cyberpunk, action

RESULT :

Thus, The Find Tags Associated with Each Movie with Hadoop Map-Reduce is successfully executed.

Question 11 :

IMPLEMENT K-MEANS CLUSTERING ALGORITHM USING HADOOP MAP-REDUCE

AIM :

To Implement K-Means Clustering Algorithm Using Hadoop Map-reduce.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop
Step - 2 :- Download a text file ‘ centroids.txt ’ from GitHub
Step - 3 :- Download a text file ‘ points.txt ’ from GitHub
Step - 4 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]  %%writefile mapper.py
     #!/usr/bin/env python

     import sys
     import numpy as np

     centroids = np.loadtxt('centroids.txt', delimiter=',')

     euclidean_distance = lambda point1, point2: np.sqrt(np.sum((point1 - point2) ** 2))

     for line in sys.stdin:
         point = np.array([float(field) for field in line.strip().split(',')])
         closest_centroid = min(range(len(centroids)), key=lambda i:
         euclidean_distance(point, centroids[i]))

         print(f'{closest_centroid}\t{",".join(map(str, point))}')
```

Output :-

Writing mapper.py

Step - 5 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
[ ]  %%writefile reducer.py
     #!/usr/bin/env python

     import sys
     import numpy as np

     centroids = {}

     for line in sys.stdin:
         centroid_id, point_str = line.strip().split('\t', 1)
         point = np.array(list(map(float, point_str.split(','))))
         centroids.setdefault(centroid_id, []).append(point)
```

```
print("Cluster ID\tCluster Centroid(X,Y)")
print()

for centroid_id, points in centroids.items():
    new_centroid = np.mean(points, axis=0)
                print(f'{centroid_id}\t\t{",".join(map(str, new_centroid))}')
```

Output :-

Writing reducer.py

Step - 6 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 7 :- Move text File from ‘ points.txt ’ to ‘ input/ ’

Command :-

```
[ ]    !hdfs dfs -mv  points.txt  input/
```

Output :-

text File is moved

Step - 8 :- Display ‘ centroids.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat centroids.txt
```

Output :-

1.0,2.0
5.0,6.0
10.0,11.0

Step - 9 :- Display ‘ points.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/points.txt
```

Output :-

1.0,2.0
2.0,3.0

3.0,4.0
4.0,5.0
5.0,6.0
6.0,7.0
7.0,8.0
8.0,9.0
9.0,10.0
10.0,11.0
11.0,12.0
12.0,13.0
13.0,14.0
14.0,15.0
15.0,16.0

Step - 10 :- Run to K-Means Clustering Algorithm through hadoop mapreduce

Command :-

```
[ ]  !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
       -files mapper.py,reducer.py \
       -mapper mapper.py \
       -reducer reducer.py \
       -input input \
       -output output
```

Output :-

Step - 11 :- Display output for Map Reduced the Center of the three Clusters

Command :-

```
[ ]  !cat /content/output/part-00000
```

Output :-

```
Cluster ID      Cluster Centroid(X,Y)

0               2.0,3.0
1               5.5,6.5
2               11.5,12.5
```

RESULT :

Thus, The Implement K-Means Clustering Algorithm Using Hadoop Map-reduce is successfully executed.

Question 12 :

FIND THE NUMBER OF PRODUCTS SOLD IN EACH COUNTRY USING MAP-REDUCE

AIM :

To Find The Number of Products Sold in Each Country by Considering Sales Data Containing Field Like Tranction_Date, Product, Price, Name, City, State, Country, Account_Created, Last_Login, Latitude, Longitude .

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop

Step - 2 :- Download a text file ‘ sales_data.txt ’ from GitHub

Step - 3 :- Create a python file ‘ mapper.py ’ and add program

Command :-

```
[ ]   %%writefile mapper.py
      #!/usr/bin/env python

      import sys

      for line in sys.stdin:
          fields = line.strip().split(',')
          if len(fields) >= 7:
              print('%s\t%s' % (fields[1], fields[6]))
```

Output :-

Writing mapper.py

Step - 4 :- Create a python file ‘ reducer.py’ and add program

Command :-

```
[ ]   %%writefile reducer.py
      #!/usr/bin/env python

      import sys

      current_product = None
      current_country = None
      current_count = 0

      print('{:15} {:<16} {:<15}'.format("product","Country","Count"))
      print()

      for line in sys.stdin:
          line = line.strip()
          product, country = line.split('\t', 1)
```

```
            if current_product == product and current_country == country:
                current_count += 1
            else:
                if current_product:
                    print(f"{current_product:<15} {current_country:<16} {current_count}")
                current_count = 1
                current_product, current_country = product, country

        if current_product:
            print(f"{current_product:<15} {current_country:<16} {current_count}")
```

Output :-

Writing reducer.py

Step - 5 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 6 :- Move text File from ‘ sales_data.txt ’ to ‘ input/ ’

Command :-

```
[ ]    !hdfs dfs -mv  sales_data.txt   input/
```

Output :-

text File is moved

Step - 7 :- Display ‘ sales_data.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/sales_data.txt
```

Output :-

```
2024-04-01,Product_A,50,John,Dallas,Texas,USA,2023-01-01,2024-03-30,32.7767,-96.7970
2024-04-01,Product_B,30,Jane,New York,New York,USA,2022-06-15,2024-03-31,40.7128,-74.0060
2024-04-02,Product_A,50,John,Los Angeles,California,USA,2021-12-10,2024-04-01,34.0522,-118.2437
2024-04-02,Product_C,20,Alice,Paris,,France,2023-05-20,2024-03-29,48.8566,2.3522
2024-04-03,Product_A,50,John,London,,UK,2023-02-14,2024-04-02,51.5074,-0.1278
2024-04-03,Product_B,30,Bob,Sydney,NSW,Australia,2022-09-30,2024-04-01,-33.8688,151.2093
2024-04-03,Product_A,50,John,New Delhi,,India,2023-04-05,2024-03-31,28.6139,77.2090
2024-04-04,Product_B,30,Bob,Tokyo,,Japan,2022-12-25,2024-03-30,35.6895,139.6917
2024-04-05,Product_A,50,John,Beijing,,China,2023-01-20,2024-04-02,39.9042,116.4074
2024-04-05,Product_C,20,Emma,Berlin,,Germany,2022-08-15,2024-03-29,52.5200,13.4050
2024-04-06,Product_B,30,Bob,Rio de Janeiro,,Brazil,2022-11-10,2024-04-02,-22.9068,-43.1729
2024-04-07,Product_A,50,John,Moscow,,Russia,2023-03-03,2024-04-01,55.7558,37.6176
```

2024-04-08,Product_C,20,Emma,Mexico City,,Mexico,2022-07-05,2024-03-30,19.4326,-99.1332
2024-04-09,Product_B,30,Bob,Istanbul,,Turkey,2022-10-01,2024-04-03,41.0082,28.9784
2024-04-10,Product_A,50,John,Cairo,,Egypt,2023-06-10,2024-04-01,30.0444,31.2357
2024-04-11,Product_B,30,Bob,Buenos Aires,,Argentina,2023-01-30,2024-03-31,-34.6037,-58.3816
2024-04-12,Product_A,50,John,Lima,,Peru,2023-04-20,2024-04-03,-12.0464,-77.0428
2024-04-13,Product_C,20,Emma,Bangkok,,Thailand,2022-12-01,2024-04-02,13.7563,100.5018
2024-04-14,Product_B,30,Bob,Seoul,,South Korea,2023-02-14,2024-03-29,37.5665,126.9780
2024-04-15,Product_A,50,John,Mumbai,,India,2023-07-10,2024-04-01,19.0760,72.8777
2024-04-15,Product_B,30,Bob,Lagos,,Nigeria,2023-03-15,2024-04-03,6.5244,3.3792
2024-04-16,Product_C,20,Emma,Kinshasa,,DR Congo,2023-01-10,2024-04-02,-4.4419,15.2663
2024-04-17,Product_A,50,John,Karachi,,Pakistan,2023-08-25,2024-04-01,24.8607,67.0011
2024-04-18,Product_B,30,Bob,Shanghai,,China,2023-04-30,2024-04-03,31.2304,121.4737
2024-04-19,Product_A,50,John,Moscow,,Russia,2023-06-01,2024-04-01,55.7558,37.6176
2024-04-20,Product_C,20,Emma,Beijing,,China,2023-02-18,2024-03-29,39.9042,116.4074
2024-04-21,Product_B,30,Bob,Berlin,,Germany,2023-01-15,2024-04-02,52.5200,13.4050
2024-04-22,Product_A,50,John,Rio de Janeiro,,Brazil,2023-05-05,2024-04-01,-22.9068,-43.1729
2024-04-23,Product_C,20,Emma,Mexico City,,Mexico,2023-03-20,2024-04-03,19.4326,-99.1332
2024-04-24,Product_B,30,Bob,Tokyo,,Japan,2023-07-30,2024-04-01,35.6895,139.6917
2024-04-25,Product_A,50,John,Paris,,France,2023-10-10,2024-04-02,48.8566,2.3522
2024-04-26,Product_C,20,Emma,Sydney,NSW,Australia,2023-04-25,2024-04-03,-33.8688,151.2093
2024-04-27,Product_B,30,Bob,London,,UK,2023-01-05,2024-04-01,51.5074,-0.1278
2024-04-28,Product_A,50,John,New York,New York,USA,2022-09-01,2024-04-02,40.7128,-74.0060
2024-04-29,Product_B,30,Bob,Dallas,Texas,USA,2022-05-20,2024-04-03,32.7767,-96.7970
2024-04-30,Product_C,20,Emma,Los Angeles,California,USA,2023-03-10,2024-04-01,34.0522,-118.2437

Step - 8 :- Run to Find The Number of Products Sold in Each Country through hadoop mapreduce

Command :-

```
[ ]  !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
       -files mapper.py,reducer.py \
       -mapper mapper.py \
       -reducer reducer.py \
       -input input \
       -output output
```

Output :-

Step - 9 :- Display output for Map Reduced The Number of Products Sold in Each Country

Command :-

```
[ ]  !cat /content/output/part-00000
```

Output :-

```
product         Country          Count

Product_A       China            1
Product_A       India            1
Product_A       UK               1
Product_A       USA              2
Product_A       Brazil           1
Product_A       Russia           1
Product_A       Pakistan         1
Product_A       India            1
```

```
Product_A       USA             1
Product_A       Peru            1
Product_A       Egypt           1
Product_A       France          1
Product_A       Russia          1
Product_B       Japan           1
Product_B       USA             1
Product_B       Australia       1
Product_B       Nigeria         1
Product_B       Japan           1
Product_B       South Korea     1
Product_B       USA             1
Product_B       Turkey          1
Product_B       Germany         1
Product_B       UK              1
Product_B       Brazil          1
Product_B       China           1
Product_B       Argentina       1
Product_C       USA             1
Product_C       Australia       1
Product_C       Mexico          1
Product_C       China           1
Product_C       DR Congo        1
Product_C       Mexico          1
Product_C       Germany         1
Product_C       France          1
Product_C       Thailand        1
```

RESULT :

Thus, The Number of Products Sold in Each Country with Hadoop Map-Reduce is successfully executed.

Question 13 :

DEVELOP A MAPREDUCE PROGRAM TO FIND THE FREQUENCY OF BOOKS PUBLISHED EACHYEAR AND FIND IN WHICH YEAR MAXIMUM NUMBER OF BOOKS WERE PUBLISHED

AIM :

To Develop a MapReduce program to find the frequency of books published each year and find in which year maximum number of books were published using the following data :- Title, Author, Published year, Author country, Language, No of pages.

PROCEDURE :

Step - 1 :- Installing jdk and Hadoop

Step - 2 :- Download a text file ' books_dataset.txt ' from GitHub

Step - 3 :- Create a python file ' mapper.py ' and add program

Command :-

```
[ ]  %%writefile mapper.py
     #!/usr/bin/env python

     import sys

     for line in sys.stdin:
         print(f"{line.strip().split(',')[3]}\t1")
```

Output :-

Writing mapper.py

Step - 4 :- Create a python file ' reducer.py' and add program

Command :-

```
[ ]  %%writefile reducer.py
     #!/usr/bin/env python

     import sys

     current_year, max_year, max_count = None, None, 0

     print("Year\tCount")  # Add column names
     print()

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

```
            if current_count > max_count:
                max_count, max_year = current_count, current_year
            current_year, current_count = year, count

    if current_year:
        print(f"{current_year}\t{current_count}")

    if max_year:
        print()
        print(f"Year with maximum books published:{max_year}(no.of.books :{max_count})")
```

Output :-

Writing reducer.py

Step - 5 :- Create a new directory folder ‘ input ’

Command :-

```
[ ]    !hdfs dfs -mkdir input
```

Output:-

Directory created

Step - 6 :- Move text File from ‘ books_dataset.txt ’ to ‘ input/ ’

Command :-

```
[ ]    !hdfs dfs -mv  books_dataset.txt   input/
```

Output :-

text File is moved

Step - 7 :- Display ‘ books_dataset.txt ’ File content

Command :-

```
[ ]    !hdfs dfs -cat input/books_dataset.txt
```

Output :-

1,Agree.,Jeffrey Wright,2017,Spain,Spanish,269
2,Camera result.,Richard Chandler,2018,Greece,German,884
3,Opportunity left us.,Nathan Nelson,2021,Senegal,English,189
4,Team I.,Renee Yang,2010,Cameroon,French,513
5,Note represent.,Nathan Joyce,2020,Qatar,French,861
6,Focus suddenly past.,Suzanne Dawson,2016,Malawi,Italian,753
7,Boy bad.,Rachel Mclean,2023,Dominica,Italian,583
8,Reduce bar may resource.,Jennifer Moore,2024,Saint Pierre and Miquelon,German,364
9,On see join.,Denise Hanson,2019,Belarus,Spanish,380
10,Soon front include.,Todd Sanchez,2023,Zimbabwe,Spanish,334
11,Expert us.,Katelyn Alvarado,2024,Yemen,Spanish,121
12,Think specific system.,Jill Soto,2014,Turks and Caicos Islands,Spanish,926
13,Understand Congress.,Hayley Johnson,2019,Cote d'Ivoire,Spanish,899

14,Simply business.,David Estrada,2013,Saint Vincent and the Grenadines,German,313
15,Take baby.,Tara Stokes,2024,Falkland Islands (Malvinas),English,426
16,Good century research.,Maxwell Green,2021,Bulgaria,Spanish,780
17,Our imagine effort.,Jacqueline White PhD,2012,Belgium,Spanish,657
18,Prove describe individual.,Stephen White,2016,United Kingdom,Italian,337
19,High.,Anthony Moore,2010,Saint Helena,Italian,987
20,Toward learn.,Larry Waters,2015,Faroe Islands,English,765
21,Image available can.,Pamela Bond,2019,Guam,Italian,498
22,Camera nor.,Jennifer Wright,2024,Guinea,Italian,695
23,These hospital apply.,Justin Anderson,2010,Saint Helena,French,60
24,Knowledge final.,Rodney Lewis,2021,Norfolk Island,German,181
25,Human ask both.,Stephen Cain,2023,British Virgin Islands,Italian,109
26,Pretty.,David Lynch,2010,French Polynesia,German,119
27,Bag nearly.,Anne Pacheco,2015,Belize,Italian,460
28,Occur knowledge science.,Bethany Doyle,2020,Spain,French,819
29,Issue sell.,Jennifer Woodward,2021,Netherlands,German,423
30,Well small near.,Mrs. Angela Arellano,2013,Samoa,German,377
31,Key can.,Kimberly Brown,2014,Nigeria,Spanish,491
32,Operation three.,Lindsay Hamilton,2021,Bhutan,Spanish,525
33,Begin.,Michael Webb,2015,French Guiana,Spanish,794
34,Area under name.,Laura Smith,2010,Aruba,English,646
35,Style without challenge.,David Jones,2010,Monaco,Italian,186
36,Personal other draw.,Julie Wood,2017,Western Sahara,French,793
37,Easy.,Paul Reynolds,2021,Afghanistan,French,830
38,Soldier fear resource.,James Gould,2017,Greenland,German,98
39,Cut similar.,Megan Torres,2017,El Salvador,French,378
40,Turn five put.,Debbie Gould,2024,Algeria,English,610
41,End former.,Marcus Herring,2021,Norfolk Island,English,853
42,See wide benefit.,Julie Paul,2014,Saudi Arabia,Italian,906
43,Day forget.,Sheila Johnson,2018,Germany,German,672
44,Deep thousand consider.,Bruce Page,2013,Swaziland,French,855
45,Staff idea simple.,Heidi Thomas,2019,Guernsey,German,751
46,Continue no.,Stephanie Franklin,2014,Mauritania,English,63
47,Morning company benefit.,Kathleen Lopez,2012,Switzerland,Spanish,684
48,Member nice.,Richard Wallace DDS,2016,Denmark,French,426
49,Consider early.,Greg Harris,2014,Belarus,German,841
50,Foreign include bag.,Christopher Lara,2024,Bhutan,German,341

Step - 8 :- Run To Find in Which Year Maximum Number of Books Were Published Through Hadoop Mapreduce

Command :-

```
[ ]  !hadoop jar /content/hadoop-3.3.6/share/hadoop/tools/lib/hadoop-streaming-3.3.6.jar \
       -files mapper.py,reducer.py \
       -mapper mapper.py \
       -reducer reducer.py \
       -input input \
       -output output
```

Output :-

Step - 9 :- Display Output for Map Reduced Which Year Maximum Number of Books were Published

Command :-

```
[ ]  !cat /content/output/part-00000
```

Output :-

```
Year    Count

2010    6
2012    2
2013    3
2014    5
2015    3
2016    3
2017    4
2018    2
2019    4
2020    2
2021    7
2023    3
2024    6

Year with maximum books published: 2021 (no.of.books : 7)
```

RESULT :

Thus, The find in which year maximum number of books were published with Hadoop Map-Reduce is successfully executed.