

# **CSL7110: Big Data Frameworks**

## Assignment Report

**Name:** Ravi Sharma  
**Roll Number:** M25CSA024  
**Course:** CSL7110  
**Deadline:** February 14, 2026

## **GitHub Repository**

The complete source code for this assignment is available at: [https://github.com/Ravi110296/CSL7110\\_Assignment/blob/main](https://github.com/Ravi110296/CSL7110_Assignment/blob/main)

# 1 Apache Hadoop and MapReduce

## 1.1 Question 1: Running WordCount Example

The WordCount example provided by Apache Hadoop was executed successfully to verify the correct setup of the Hadoop environment.

### Snapshots

```
root@LAPTOP-CPCUQW8J:~# hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar wordcount /user/root/input /user/root/output
2026-02-06 19:59:28.329 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2026-02-06 19:59:28.446 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2026-02-06 19:59:28.446 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2026-02-06 19:59:28.711 INFO input.FileInputFormat: Total input files to process : 1
2026-02-06 19:59:28.730 INFO mapreduce.JobSubmitter: number of splits:1
2026-02-06 19:59:28.967 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local905053919_0001
2026-02-06 19:59:28.967 INFO mapreduce.JobSubmitter: Executing with tokens: []
2026-02-06 19:59:29.032 INFO mapreduce.Job: The url to track the job: http://localhost:8088/
2026-02-06 19:59:29.034 INFO mapreduce.Job: Running job: job_local905053919_0001
2026-02-06 19:59:29.037 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2026-02-06 19:59:29.051 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2026-02-06 19:59:29.051 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2026-02-06 19:59:29.051 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2026-02-06 19:59:29.053 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitter
2026-02-06 19:59:29.105 INFO mapred.LocalJobRunner: Waiting for map tasks
2026-02-06 19:59:29.105 INFO mapred.LocalJobRunner: Starting task: attempt_local905053919_0001_m_000000_0
2026-02-06 19:59:29.137 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2026-02-06 19:59:29.137 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2026-02-06 19:59:29.137 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2026-02-06 19:59:29.164 INFO mapred.Task: Using ResourceCalculatorProcessTree : []
2026-02-06 19:59:29.169 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/user/root/input/sample.txt:0+53
2026-02-06 19:59:29.301 INFO mapred.MapTask: (EQUATOR) 0 kvi 26214396(104857584)
2026-02-06 19:59:29.301 INFO mapred.MapTask: mapreduce.task.io.sort.mb: 100
2026-02-06 19:59:29.301 INFO mapred.MapTask: soft limit at 83886080
2026-02-06 19:59:29.301 INFO mapred.MapTask: bufstart = 0; bufvoid = 104857600
2026-02-06 19:59:29.301 INFO mapred.MapTask: kvstart = 26214396; length = 6553600
2026-02-06 19:59:29.307 INFO mapred.MapTask: InputSplit bytes=53
2026-02-06 19:59:29.307 INFO mapred.MapTask: InputSplit collect class = org.apache.hadoop.mapred.MapTask$MapOutputBuffer
2026-02-06 19:59:29.416 INFO mapred.LocalJobRunner: mapred.MapTask: Starting flush of map output
2026-02-06 19:59:29.418 INFO mapred.MapTask: Spilling map output
2026-02-06 19:59:29.418 INFO mapred.MapTask: mapred.MapTask: bufstart = 0; bufend = 105; bufvoid = 104857600
2026-02-06 19:59:29.418 INFO mapred.MapTask: kvstart = 26214396(104857584); kend = 26214248(104857392); length = 49/6553600
2026-02-06 19:59:29.432 INFO mapred.MapTask: Finished spill 0
2026-02-06 19:59:29.432 INFO mapred.Task: Task:attempt_local905053919_0001_m_000000_0 is done. And is in the process of committing
2026-02-06 19:59:29.441 INFO mapred.LocalJobRunner: map
2026-02-06 19:59:29.449 INFO mapred.Task: Task 'attempt_local905053919_0001_m_000000_0' done.
2026-02-06 19:59:29.457 INFO mapred.Task: Final Counters for attempt_local905053919_0001_m_000000_0: Counters: 24
File System Counters
```

Figure 1: Running WordCount MapReduce job

```
2026-02-06 19:59:29.457 INFO mapred.Task: Final Counters for attempt_local905053919_0001_m_000000_0: Counters: 24
File System Counters
  FILE: Number of bytes read=281533
  FILE: Number of bytes written=917318
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
HDFS: Number of bytes read=53
HDFS: Number of bytes written=8
HDFS: Number of read operations=5
HDFS: Number of large read operations=0
HDFS: Number of write operations=1
HDFS: Number of bytes read erasure-coded=0
Map-Reduce Framework
  Map input records=2
  Map output records=13
  Map output bytes=105
  Map output materialized bytes=87
  Input split bytes=113
  Combine input records=13
  Combine output records=8
  Spilled Records=8
  Failed Shuffles=0
  Merged Map outputs=0
  GC time elapsed (ms)=4
  Total committed heap usage (bytes)=265289728
File Input Format Counters
  Bytes Read=53
2026-02-06 19:59:29.457 INFO mapred.LocalJobRunner: Finishing task: attempt_local905053919_0001_m_000000_0
2026-02-06 19:59:29.458 INFO mapred.LocalJobRunner: map task executor complete.
2026-02-06 19:59:29.462 INFO mapred.LocalJobRunner: Waiting for reduce tasks
2026-02-06 19:59:29.462 INFO mapred.LocalJobRunner: Starting task: attempt_local905053919_0001_r_000000_0
2026-02-06 19:59:29.472 INFO output.PathOutputCommitterFactory: No output committer factory defined, defaulting to FileOutputCommitterFactory
2026-02-06 19:59:29.472 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2026-02-06 19:59:29.472 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output directory:false, ignore cleanup failures: false
2026-02-06 19:59:29.473 INFO mapred.Task: Using ResourceCalculatorProcessTree : []
2026-02-06 19:59:29.477 INFO mapred.ReduceTask: Using ShuffleConsumerPlugin: org.apache.hadoop.mapreduce.task.reduce.Shuffle@71cf769f
2026-02-06 19:59:29.479 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
2026-02-06 19:59:29.497 INFO reduce.MergeManagerImpl: MergerManager: memoryLimit=1441582208, maxSingleShuffleLimit=360395552, mergeThreshold=951444288, ioSortFactor=10, memToMemMergeOutputsThreshold=10
```

Figure 2: WordCount in progress

```

2026-02-06 19:59:29,530 INFO reduce.LocalFetcher: localfetcher#1 about to shuffle output of map attempt_local905053919_0001_m_000000_0 decomp: 83 len: 87 to
MEMORY
2026-02-06 19:59:29,534 INFO reduce.InMemoryMapOutput: Read 83 bytes from map-output for attempt_local905053919_0001_m_000000_0
2026-02-06 19:59:29,537 INFO reduce.MergeManagerImpl: closeInMemoryFile -> map-output of size: 83, inMemoryMapOutputs.size() -> 1, commitMemory -> 0, usedMemory ->83
2026-02-06 19:59:29,540 INFO reduce.EventFetcher: EventFetcher is interrupted.. Returning
2026-02-06 19:59:29,541 INFO mapred.LocalJobRunner: 1 / 1 copied.
2026-02-06 19:59:29,541 INFO reduce.MergeManagerImpl: finalMerge called with 1 in-memory map-outputs and 0 on-disk map-outputs
2026-02-06 19:59:29,541 INFO mapred.Merger: Merging 1 sorted segments
2026-02-06 19:59:29,541 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 77 bytes
2026-02-06 19:59:29,551 INFO reduce.MergeManagerImpl: Merged 1 segments, 83 bytes to disk to satisfy reduce memory limit
2026-02-06 19:59:29,551 INFO reduce.MergeManagerImpl: Merging 1 files, 87 bytes from disk
2026-02-06 19:59:29,551 INFO reduce.MergeManagerImpl: Merging 0 segments, 0 bytes from memory into reduce
2026-02-06 19:59:29,551 INFO reduce.MergeManagerImpl: Merging 1 sorted segments
2026-02-06 19:59:29,551 INFO mapred.Merger: Down to the last merge-pass, with 1 segments left of total size: 77 bytes
2026-02-06 19:59:29,551 INFO mapred.LocalJobRunner: 1 / 1 copied.
2026-02-06 19:59:29,551 INFO mapred.Merger: skip.on is deprecated. Instead, use mapreduce.job.skiprecords
2026-02-06 19:59:29,551 INFO mapred.Task: Task attempt_local905053919_0001_r_000000_0 is done. And is in the process of committing
2026-02-06 19:59:29,551 INFO mapred.LocalJobRunner: 1 / 1 copied
2026-02-06 19:59:29,551 INFO mapred.Task: Task attempt_local905053919_0001_r_000000_0 is allowed to commit now
2026-02-06 19:59:29,653 INFO output.FileOutputCommitter: Saved output of task 'attempt_local905053919_0001_r_000000_0' to hdfs://localhost:9000/user/root/output
2026-02-06 19:59:29,654 INFO mapred.LocalJobRunner: reduce > reduce
2026-02-06 19:59:29,654 INFO mapred.Task: Task 'attempt_local905053919_0001_r_000000_0' done.
2026-02-06 19:59:29,655 INFO mapred.Task: Final Counters for attempt_local905053919_0001_r_000000_0: Counters: 30
    File System Counters
        FILE: Number of bytes read=281739
        FILE: Number of bytes written=917405
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=83
        HDFS: Number of bytes written=49
        HDFS: Number of read operations=10
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=3
        HDFS: Number of bytes read erasure-coded=0
    Map-Reduce Framework
        Combine input records=0
        Combine output records=0
        Reduce input groups=8

```

Figure 3: WordCount in progress

```

Reduce input records=8
Reduce output records=8
Spilled Records=8
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=0
Total committed heap usage (bytes)=265289728
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Output Format Counters
  Bytes Written=49
2026-02-06 19:59:29,655 INFO mapred.LocalJobRunner: Finishing task: attempt_local905053919_0001_r_000000_0
2026-02-06 19:59:29,656 INFO mapred.LocalJobRunner: reduce task executor complete.
2026-02-06 19:59:30,050 INFO mapreduce.Job: Job job_local905053919_0001 running in uber mode : false
2026-02-06 19:59:30,050 INFO mapreduce.Job: map 100% reduce 100%
2026-02-06 19:59:30,057 INFO mapreduce.Job: Job job_local905053919_0001 completed successfully
2026-02-06 19:59:30,068 INFO mapreduce.Job: Counters: 36
    File System Counters
        FILE: Number of bytes read=56272
        FILE: Number of bytes written=1834723
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=106
        HDFS: Number of bytes written=49
        HDFS: Number of read operations=15
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=4
        HDFS: Number of bytes read erasure-coded=0
    Map-Reduce Framework
        Map input records=2
        Map output records=13
        Map output bytes=105
        Map output materialized bytes=87
        Input split bytes=113

```

Figure 4: WordCount in progress

```

Combine output records=8
Reduce input groups=8
Reduce shuffle bytes=87
Reduce input records=8
Reduce output records=8
Spilled Records=16
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=4
Total committed heap usage (bytes)=530579456
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=53
File Output Format Counters
  Bytes Written=49

```

Figure 5: WordCount in progress

```
root@LAPTOP-CPCUQN0J:~# hdfs dfs -ls /user/root/output
Found 2 items
-rw-r--r--  1 root supergroup          0 2026-02-06 19:59 /user/root/output/_SUCCESS
-rw-r--r--  1 root supergroup        49 2026-02-06 19:59 /user/root/output/part-r-00000
```

Figure 6: Reducer writing final WordCount output to HDFS

```
root@LAPTOP-CPCUQN0J:~# hdfs dfs -cat /user/root/output/part-r-00000
all      2
are      2
get      1
lucky    1
night    2
to       1
up       2
we       2
```

Figure 7: WordCount output in HDFS

## 1.2 Question 2: Mapper Input and Output

During the Map phase of the WordCount program, each line of input text is tokenized into individual words. For each occurrence of a word, the mapper emits a key–value pair where the key is the word and the value is the integer 1. ("we're", 1) ("up", 1) ("all", 1) ("night", 1) ("till", 1) ("the", 1) ("sun", 1) ...

Map input key type: LongWritable

Map input value type: Text

Map output key type: Text

Map output value type: IntWritable.

## 1.3 Question 3: Map Output Key–Value Pairs

After the Map phase finishes, Hadoop performs:

Shuffle: groups all values by key (word)

Sort: keys are sorted alphabetically

So all values for the same word are collected together.

so the input looks like ("we're", [1, 1, 1, 1]) ("up", [1, 1, 1, 1]) ("all", [1, 1, 1, 1]) ("night", [1, 1, 1, 1]) ("till", [1, 1, 1, 1]) ("the", [1, 1, 1, 1]) ("sun", [1, 1, 1, 1])

the types of key and values of input of reduce is Key: Text Value: Iterable[IntWritable]; while that of output is Key: Text Value: IntWritable

## 1.4 Question 4: Shuffle and Sort Phase

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

@Override
public void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {

public static class Reduce
    extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override
public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
```

In the WordCount program, the Mapper class extends Mapper`<LongWritable, Text, Text, IntWritable>`, where the input key is the byte offset of each line and the input value is the line of text. The mapper emits intermediate key–value pairs of type (Text, IntWritable).

The Reducer class extends Reducer`<Text, IntWritable, Text, IntWritable>`, where each key corresponds to a word and the values represent the counts emitted by the mapper. The reducer aggregates these values to produce the final word counts.

The output key and value classes of the job are set using Text.class and IntWritable.class.

## 1.5 Question 5: Reduce Phase

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

String line = value.toString().toLowerCase();

// Remove punctuation
line = line.replaceAll("[^a-zA-Z ]", " ");

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {
    Text word = new Text(tokenizer.nextToken());
    context.write(word, new IntWritable(1));
}
}
```

The map() function processes one line of input at a time. The input value is first converted to lowercase to ensure case-insensitive counting. Punctuation is removed using the replaceAll() method with a regular expression. The cleaned line is then tokenized into individual words using a StringTokenizer. For each word, the mapper emits an intermediate key–value pair (word, 1) indicating one occurrence of the word.

## 1.6 Question 6: Error Detection

```
public void reduce(Text key, Iterable<IntWritable> values, Context context)
    throws IOException, InterruptedException {

    int sum = 0;

    for (IntWritable value : values) {
        sum += value.get();
    }

    context.write(key, new IntWritable(sum));
}
```

The reduce() function receives a key and a list of values associated with that key. Each key represents a unique word, and the values represent the counts emitted by the mapper. The reducer iterates over the list of values and computes their sum. Finally, it emits the word along with its total count as the output.

## 1.7 Question 7: WordCount on Large File (200.txt)

WordCount was executed on a larger dataset (200.txt) to analyze performance on larger input sizes.

### Snapshots

```
root@LAPTOP-CPCUQN0J:~# hdfs dfs -ls /user/root/input
Found 2 items
-rw-r--r-- 1 root supergroup 8312639 2026-02-07 00:33 /user/root/input/
200.txt
-rw-r--r-- 1 root supergroup 53 2026-02-06 19:55 /user/root/input/
sample.txt
```

Figure 8: Uploading 200.txt to HDFS

```
2026-02-07 00:36:48,825 INFO mapred.LocalJobRunner: Finishing task: attempt_local656342458_0001_r_000000_0
2026-02-07 00:36:48,826 INFO mapred.LocalJobRunner: reduce task executor complete.
2026-02-07 00:36:49,798 INFO mapreduce.Job: map 100% reduce 100%
2026-02-07 00:36:49,798 INFO mapreduce.Job: Job job_local656342458_0001 completed successfully
2026-02-07 00:36:49,811 INFO mapreduce.Job: Counters: 36
  File System Counters
    FILE: Number of bytes read=4804256
    FILE: Number of bytes written=8196247
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=16625278
    HDFS: Number of bytes written=1574586
    HDFS: Number of read operations=15
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=4
    HDFS: Number of bytes read erasure-coded=0
  Map-Reduce Framework
    Map input records=146933
    Map output records=1348566
    Map output bytes=13455246
    Map output materialized bytes=2120579
    Input split bytes=110
    Combine input records=1348566
    Combine output records=139630
    Reduce input groups=139630
    Reduce shuffle bytes=2120579
    Reduce input records=139630
    Reduce output records=139630
    Spilled Records=279266
    Shuffled Maps =1
    Failed Shuffles=0
    Merged Map outputs=1
    GC time elapsed (ms)=40
    Total committed heap usage (bytes)=525336576
  Shuffle Errors
    BAD_ID=0
    CONNECTION=0
    IO_ERROR=0
```

Figure 9: WordCount MapReduce job on 200.txt

```

        WRONG_LENGTH=0
        WRONG_MAP=0
        WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=8312639
File Output Format Counters
    Bytes Written=1574586

```

Figure 10: WordCount MapReduce job on 200.txt

```

root@LAPTOP-CPCUQNOJ:~# hdfs dfs -ls /user/root/output_200
Found 2 items
-rw-r--r--  1 root supergroup          0 2026-02-07 00:36 /user/root/output_200/_SUCCESS
-rw-r--r--  1 root supergroup  1574586 2026-02-07 00:36 /user/root/output_200/part-r-00000

```

Figure 11: storing the wordcount output to output\_200.txt

```

root@LAPTOP-CPCUQNOJ:~# head output_200.txt
      1
      1
!!!Remember.      1
"            34
"100."      1
"A            1
"Alabama,"      1
"Albemarle"      2
"Albemarle,"      1
"Alceste,"      2

```

Figure 12: Merged WordCount output

## 1.8 Question 8: Replication Factor

Why don't directories have a replication factor?

In HDFS, replication is applied only to files because files are split into data blocks that are stored on DataNodes. Directories contain only metadata such as file names and permissions and do not store actual data blocks. Since there are no blocks to replicate, directories do not have a replication factor.

How does changing the replication factor impact performance?

Increasing the replication factor improves fault tolerance and read performance because data can be accessed from multiple DataNodes. However, it increases storage usage and slows down write operations due to additional replicas. Decreasing the replication factor reduces storage and write overhead but lowers fault tolerance.

Why does changing the replication factor affect performance?

Changing the replication factor affects performance because it determines how many copies of each data block are stored across the cluster. More replicas increase network and disk

usage during writes but enable parallel reads and higher reliability, while fewer replicas reduce overhead but increase the risk of data loss.

### 1.9 Question 9: Effect of Input Split Size

In this experiment, the WordCount program was executed twice on the same input dataset. In the first run, the default input split size was used. In the second run, the input split size was increased to 256 MB using the configuration property `mapreduce.inputformat.split.maxsize`.

Increasing the input split size reduces the number of input splits, which in turn reduces the number of mapper tasks created by Hadoop. As a result, the job execution time changes due to reduced task initialization and scheduling overhead. For small datasets, the difference in execution time is minimal, while for larger datasets, fewer mappers can lead to improved performance by lowering overhead.

The execution times observed in both runs demonstrate how input split size influences the parallelism and performance of a MapReduce job.

#### Snapshots

```
root@LAPTOP-CPCUQN0J:~# /usr/bin/time hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar wordcount /user/root/input /user/root/output_time
```

Figure 13: Running WordCount without input split size constraint

```
Bytes Written: 1074000  
15.15user 2.09system 0:08.69elapsed 198%CPU (0avgtext+0avgdata 396656maxresident)k  
45288inputs+11960outputs (28major+110621minor)pagefaults 0swaps
```

Figure 14: Execution time and CPU usage without split optimization

```
root@LAPTOP-CPCUQN0J:~# /usr/bin/time hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-3.3.6.jar wordcount \  
-D mapreduce.input.format.split.maxsize=268435456 \
```

Figure 15: Running WordCount with custom input split size

```
WRONG_REDUCE=0  
File Input Format Counters  
    Bytes Read=8312639  
File Output Format Counters  
    Bytes Written=1574586  
9.88user 1.61system 0:05.39elapsed 213%CPU (0avgtext+0avgdata 399100maxresident)k  
96880inputs+11960outputs (131major+110850minor)pagefaults 0swaps
```

Figure 16: Execution time comparison showing performance difference

## 2 PySpark Experiments

\*Due to the large size of the complete Project Gutenberg corpus, a representative subset of Gutenberg texts was used for analysis. This subset preserves the original Gutenberg metadata format and is sufficient to demonstrate metadata extraction, TF-IDF similarity computation, and author influence analysis.

### 2.1 Question 10: Metadata Extraction

Metadata fields such as language appear only once per book in Project Gutenberg texts. Since the dataset was loaded line-by-line, most rows do not contain metadata values. Therefore,

rows with non-empty metadata fields were filtered and deduplicated by file name to obtain one metadata record per book before analysis.

To find the most common language, metadata rows containing non-empty language values were filtered and deduplicated per book. The dataset was then grouped by language and ordered by count in descending order. English was found to be the most common language among the analyzed books.

## Snapshots

```
root@LAPTOP-CPCUQN0J:~/gutenberg# ls
11-0.txt 1342-0.txt 1661-0.txt 84-0.txt 98-0.txt
root@LAPTOP-CPCUQN0J:~/gutenberg# |
```

Figure 17: Loading Gutenberg dataset

```
>>> books_df.printSchema()
root
|-- text: string (nullable = true)
|-- file_name: string (nullable = false)
```

Figure 18: Inspecting DataFrame Schema

```
>>> books_df.show(5, truncate=False)
+-----+-----+
|text          |file_name   |
+-----+-----+
|The Project Gutenberg eBook of A Tale of Two Cities, by Charles Dickens |file:///root/gutenberg/98-0.txt|
|This eBook is for the use of anyone anywhere in the United States and |file:///root/gutenberg/98-0.txt|
|most other parts of the world at no cost and with almost no restrictions|file:///root/gutenberg/98-0.txt|
|whatsoever. You may copy it, give it away or re-use it under the terms |file:///root/gutenberg/98-0.txt|
+-----+-----+
only showing top 5 rows
```

Figure 19: Previewing Loaded Text Data

```
>>> from pyspark.sql.functions import regexp_extract
>>> metadata_df = books_df \
...     .withColumn("title", ... .withColumn("text", regexp_extract("text", r>Title:\s*(.*)", 1)) \ 
...     .withColumn("author", ... .withColumn("text", regexp_extract("text", r"Author:\s*(.*)", 1)) \ 
...     .withColumn("release_date", ... .withColumn("text", regexp_extract("text", r"Release Date:\s*(.*)", 1)) \ 
...     .withColumn("language", ... .withColumn("text", regexp_extract("text", r"Language:\s*(.*)", 1)) \ 
...     .withColumn("encoding", ... .withColumn("text", regexp_extract("text", r"Character set encoding:\s*(.*)", 1)))
>>>
>>> metadata_df_filtered = metadata_df.filter(
...     (metadata_df.title != "") | (metadata_df.author != ""))
... )
>>> metadata_df_filtered.select(
...     "file_name", "title", "author", "release_date", "language", "encoding"
... ).show(10, truncate=False)
+-----+-----+-----+-----+-----+
|file_name      |title           |author          |release_date|language|encoding|
+-----+-----+-----+-----+-----+
|file:///root/gutenberg/98-0.txt |A Tale of Two Cities |Charles Dickens |          |        |
|file:///root/gutenberg/98-0.txt |The Adventures of Sherlock Holmes |Arthur Conan Doyle |          |        |
|file:///root/gutenberg/1661-0.txt |The Adventures of Sherlock Holmes |Arthur Conan Doyle |          |        |
|file:///root/gutenberg/1661-0.txt |The Adventures of Sherlock Holmes |Arthur Conan Doyle |          |        |
+-----+-----+-----+-----+-----+
```

Figure 20: Metadata Extraction Using Regular Expressions

```
>>> books_per_year.show(truncate=False)
+---+---+
|year|count|
+---+---+
|1994|1    |
|2002|1    |
+---+---+
```

Figure 21: number of books released each year

```
>>> language_per_book.groupBy("language") \
rBy("count", asc... .count() \
...     .orderBy("count", ascending=False) \
...     .show(1, truncate=False)
+-----+---+
|language|count|
+-----+---+
|English |2    |
+-----+---+
```

Figure 22: Most common language

```
>>> from pyspark.sql.functions import length, avg
red.select(avg(length("title"))).show()
>>>
>>> metadata_df_filtered.select(avg(length("title"))).show()
+-----+
|avg(length(title))|
+-----+
|          13.25|
+-----+
```

Figure 23: Average title length

## 2.2 Question 11: TF-IDF and Cosine Similarity

TF-IDF vectors were computed for each book, and cosine similarity was used to identify similar documents.

## Snapshots

```
>>> from pyspark.sql.functions import lower, regexp_replace, split, explode
>>> clean_df = books_df.withColumn(
lace(lower("text...      "clean_text",
...      regexp_replace(lower("text"), "[^a-z\\s]", ""))
... )
>>> words_df = clean_df.withColumn(
explode(split("c...      "word",
...      explode(split("clean_text", "\\s+"))
... )
>>> words_df = words_df.filter(words_df.word != "")
>>> words_df.select("file_name", "word").show(10, truncate=False)
[Stage 3:>
+-----+-----+
|file_name          |word   |
+-----+-----+
|file:///root/gutenberg/98-0.txt|the    |
|file:///root/gutenberg/98-0.txt|project|
|file:///root/gutenberg/98-0.txt|gutenberg|
|file:///root/gutenberg/98-0.txt|ebook   |
|file:///root/gutenberg/98-0.txt|of     |
|file:///root/gutenberg/98-0.txt|a      |
|file:///root/gutenberg/98-0.txt|tale   |
|file:///root/gutenberg/98-0.txt|of     |
|file:///root/gutenberg/98-0.txt|two    |
|file:///root/gutenberg/98-0.txt|cities |
+-----+-----+
only showing top 10 rows
```

Figure 24: Text cleaning and tokenization

```
>>> tf_df = words_df.groupBy("file_name", "word").count()
>>> tf_df = tf_df.withColumnRenamed("count", "tf")
>>> tf_df.show(10, truncate=False)
[Stage 4:>
+-----+-----+-----+
|file_name          |word   |tf   |
+-----+-----+-----+
|file:///root/gutenberg/98-0.txt|age    |20  |
|file:///root/gutenberg/98-0.txt|us     |114 |
|file:///root/gutenberg/98-0.txt|jaw    |2   |
|file:///root/gutenberg/98-0.txt|christian|6   |
|file:///root/gutenberg/98-0.txt|about   |169 |
|file:///root/gutenberg/98-0.txt|blunderbusses|1 |
|file:///root/gutenberg/98-0.txt|pilferer|1 |
|file:///root/gutenberg/98-0.txt|look    |140 |
|file:///root/gutenberg/98-0.txt|tinderbox|1 |
|file:///root/gutenberg/98-0.txt|quieted|1 |
+-----+-----+-----+
only showing top 10 rows
>>>
```

Figure 25: Term Frequency (TF)

```
>>> from pyspark.sql.functions import countDistinct, log
>>> num_docs = books_df.select("file_name").distinct().count()
>>> df_df = tf_df.groupBy("word").agg(countDistinct("file_name").alias("df"))
>>> idf_df = df_df.withColumn("idf", log(num_docs / df_df.df))
>>> idf_df.show(10, truncate=False)
+-----+-----+-----+
|word   |df   |idf   |
+-----+-----+-----+
|superseded|2  |0.9162907318741551 |
|connected |4  |0.22314355131420976|
|knuckled  |1  |1.6094379124341003 |
|spoiling   |1  |1.6094379124341003 |
|filing    |1  |1.6094379124341003 |
|few       |5  |0.0  |
|imitation |3  |0.5108256237659907 |
|importantno|1  |1.6094379124341003 |
|biting    |2  |0.9162907318741551 |
|still     |5  |0.0  |
+-----+-----+-----+
only showing top 10 rows
```

Figure 26: Inverse Document Frequency (IDF)

```

>>> tfidf_df = tf_df.join(idf_df, on="word")
>>> tfidf_df = tfidf_df.withColumn("tfidf", tfidf_df.tf * tfidf_df.idf)
>>> tfidf_df.select("file_name", "word", "tfidf").show(10, truncate=False)
+-----+-----+-----+
|file_name          |word      |tfidf    |
+-----+-----+-----+
|file:///root/gutenberg/1342-0.txt|superseded|0.9162907318741551|
|file:///root/gutenberg/98-0.txt|superseded|0.9162907318741551|
|file:///root/gutenberg/84-0.txt|connected |1.5620048591994684|
|file:///root/gutenberg/1661-0.txt|connected |1.5620048591994684|
|file:///root/gutenberg/1342-0.txt|connected |3.3471532697131465|
|file:///root/gutenberg/98-0.txt|connected |1.7851484105136781|
|file:///root/gutenberg/98-0.txt|knuckled  |3.2188758248682006|
|file:///root/gutenberg/98-0.txt|spoiling   |1.6094379124341003|
|file:///root/gutenberg/98-0.txt|filing     |1.6094379124341003|
|file:///root/gutenberg/11-0.txt|few        |0.0
+-----+-----+-----+
only showing top 10 rows

```

Figure 27: TF-IDF computation

```

>>> similarity_df.orderBy("cosine_similarity", ascending=False).show(10, truncate=False)
[Stage 28:>          (0 + 5) / 5][Stage 29:>          (0 + 5) [Stage 28:=====] (2 + 3)
+-----+-----+-----+-----+-----+
|book2       |book1      |dot_product |norm1      |norm2      |cosine_similarity |
+-----+-----+-----+-----+-----+
|file:///root/gutenberg/84-0.txt|file:///root/gutenberg/1342-0.txt|55174.627874063894|1298.844436778923|381.323449786551|0.1499773481353911| |
|file:///root/gutenberg/1342-0.txt|file:///root/gutenberg/84-0.txt|55174.627874063838|1298.844436778923|381.323449786551|0.1499773481353911|
|file:///root/gutenberg/98-0.txt|file:///root/gutenberg/84-0.txt|13810.374828895459|380.788350933921|381.323449786551|0.83977463774616|
|file:///root/gutenberg/84-0.txt|file:///root/gutenberg/1342-0.txt|13810.374828895459|380.788350933921|381.323449786551|0.83977463774616|
|file:///root/gutenberg/1342-0.txt|file:///root/gutenberg/1661-0.txt|13611.080417857535|380.788350933921|381.323449786551|0.83977463774616|
|file:///root/gutenberg/1661-0.txt|file:///root/gutenberg/98-0.txt|33611.087579567216|1183.784835493391|816.4537493728855|1.183.784835493391|0.0359741262866656|
|file:///root/gutenberg/1342-0.txt|file:///root/gutenberg/98-0.txt|39562.94041785865|1183.784835493392|1298.844436778923|0.27596654961981384|
|file:///root/gutenberg/98-0.txt|file:///root/gutenberg/1342-0.txt|39562.94041785865|1183.784835493392|1298.844436778923|0.27596654961981384|
|file:///root/gutenberg/1661-0.txt|file:///root/gutenberg/84-0.txt|6515.813851977589|381.323449786551|846.4537493728855|0.25546564583745955|
|file:///root/gutenberg/84-0.txt|file:///root/gutenberg/1661-0.txt|6515.813851977589|381.323449786551|846.4537493728855|0.25546564583745955|
+-----+-----+-----+-----+-----+
only showing top 10 rows

```

Figure 28: Cosine Similarity Between Documents

### 2.3 Question 12: Author Influence Network

To construct the author influence network, author–year metadata was duplicated with renamed columns to avoid ambiguity during self-joins. A directed edge was created from a source author to a target author when the source author’s publication year was earlier than the target author’s year. This temporal ordering models potential literary influence between authors.

#### Snapshots

```

>>> author_year_df.show(truncate=False)
+-----+-----+-----+
|file_name          |author      |year |
+-----+-----+-----+
|file:///root/gutenberg/1661-0.txt|Arthur Conan Doyle|2002|
|file:///root/gutenberg/98-0.txt|Charles Dickens |1994|
+-----+-----+-----+

```

Figure 29: Extracted author and year metadata

```
>>> influence_df.show(truncate=False)
+-----+-----+-----+-----+
|source_author |source_year|target_author |target_year|
+-----+-----+-----+-----+
|Charles Dickens|1994      |Arthur Conan Doyle|2002      |
+-----+-----+-----+-----+
```

Figure 30: Author influence analysis

### 3 Conclusion

This assignment provided hands-on experience with Hadoop MapReduce and Apache Spark for large-scale text analytics. The experiments demonstrated distributed storage, parallel computation, and advanced analytics.