Assignment 1

Name: Arpita Kundu, Roll no: M25DE1004

Github link: https://github.com/arpita12-bot/Machine-learning-with-Big-Data/tree/main

Apache Hadoop & MapReduce

Q-1:

Run and show the working of the WordCount example as shown on the Apache Hadoop website as well as discussed in class earlier. (No need to send code for this. Just output to show that it is working is enough).

```
arpkundu@DESKTOP-J1UA354:~$ nano wordcount_input.txt
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -mkdir -p /input
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -put wordcount_input.txt /input
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -ls /input
Found 1 items
-rw-r--r--   1 arpkundu supergroup         63 2026-02-08 22:23 /input/wordcount_input.txt
arpkundu@DESKTOP-J1UA354:~$
```

Wordcount program:

```
 GNU nano 6.2                                                    WordCount.java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }

^G Help      ^O Write Out  ^W Where Is  ^K Cut       ^T Execute   ^C Location  M-U Undo    M-A Set Mark
^X Exit      ^R Read File  ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line M-E Redo    M-6 Copy
```

Prepare input in HDFS

Ouput:

```
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -cat /wordcount/output/part
-r-00000
big     1
data    1
hadoop  1
hello   3
mapreduce       1
```

Q-2:

Input file we use with following lyrics named as wc1.txt.

```
We're up all night till the sun
We're up all night to get some
We're up all night for good fun
We're up all night to get lucky
```

The input pairs for the Map phase will be the following:

```
(0, "We're up all night to the sun")
(31, "We're up all night to get some")
(63, "We're up all night for good fun")
(95, "We're up all night to get lucky")
```

The key is the byte offset starting from the beginning of the file. While we won't need this value in Word Count, it is always passed to the Mapper by the Hadoop framework.

The byte offset is a number that can be large if there are many lines in the file.

Put the input file into HDFS using following commands:

hdfs dfs -mkdir -p /lyrics/input

hdfs dfs -put wc1.txt /lyrics/input

```
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -mkdir -p /lyrics/input
hdfs dfs -put wc1.txt /lyrics/input
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -ls /lyrics/input
Found 1 items
-rw-r--r--   1 arpkundu supergroup        129 2026-02-08 23:18 /
lyrics/input/wc1.txt
arpkundu@DESKTOP-J1UA354:~$ hadoop jar $HADOOP_HOME/share/hadoop
```

Now run wordcount:

hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-mapreduce-examples-*.jar wordcount /lyrics/input /lyrics/output

Input to Map phase:

```
(LongWritable, Text)
(0,  "We're up all night till the sun")
(31, "We're up all night to get some")
(63, "We're up all night for good fun")
(95, "We're up all night to get lucky")
```

Result :

hdfs dfs -cat /lyrics/output/part-r-00000

```
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -cat /lyrics/output/part-r-00000
We're   4
all     4
for     1
fun     1
get     2
good    1
lucky   1
night   4
some    1
sun     1
the     1
till    1
to      2
up      4
arpkundu@DESKTOP-J1UA354:~$
```

Explanation : The WordCount MapReduce program was executed on Hadoop using the given lyrics as input. The output shows correct word frequencies, confirming successful execution of the Map and Reduce phases.

1. **What will the output pairs look like(Mapper)?**

   In **WordCount**, the **Mapper**:

   - Reads **one line at a time**
   - Splits the line into words
   - Emits **(word, 1)** for each word

   Output pairs from the Map phase:

   Each word generates one key-value pair:

   ```
   (We're, 1)
   (up, 1)
   (all, 1)
   (night, 1)
   (till, 1)
   (the, 1)
   (sun, 1)
   ```

   These are the **output pairs of the Map phase**, **not reduced yet**.

2. What are the **types of keys and values** in the Map phase?

   Input to the Mapper :

Component	Type (Hadoop)	Meaning
Key	LongWritable	Byte offset of the line in the file
Value	Text	One line of input text

Example:
(LongWritable, Text)
(0, "We're up all night till the sun")

Output from the Mapper:

Component	Type (Hadoop)	Meaning
Key	Text	A word
Value	IntWritable	Count = 1

Example:

(Text, IntWritable)
("night", 1)

Q-3:

For the reduce phase, some of the output pairs will be the following:

```
("up", 4)
("to", 3)
("get", 2)
("lucky", 1)
```

1. **What will the input pairs look like?**
   Before the Reduce phase, Hadoop performs the **Shuffle and Sort** step.
   This step **groups all values with the same key** together.

   So, the reducer receives input in the form:
   (word, list_of_values)

   Reduce Phase Input Pairs:
   ("up", [1, 1, 1, 1])
   ("to", [1, 1, 1])
   ("get", [1, 1])
   ("lucky", [1])
   Each 1 represents one occurrence of the word emitted by the Mapper.

2. **What will be the types of keys and values of the input and output pairs in the Reduce phase?**

   Input to the Reducer:

Component	Hadoop Data Type	Description
Key	Text	A word
Value	Iterable<IntWritable>	List of counts

Example:

(Text, Iterable<IntWritable>)
("up", [1,1,1,1])

Output from the Reducer:

Component	Hadoop Data Type	Description
Key	Text	A word
Value	IntWritable	Total count

Example:

(Text, IntWritable)
("up", 4)

In the Reduce phase, input pairs consist of a word and an iterable list of counts, and output pairs consist of the word and its total frequency. Reducer receives grouped values and produces final word counts.

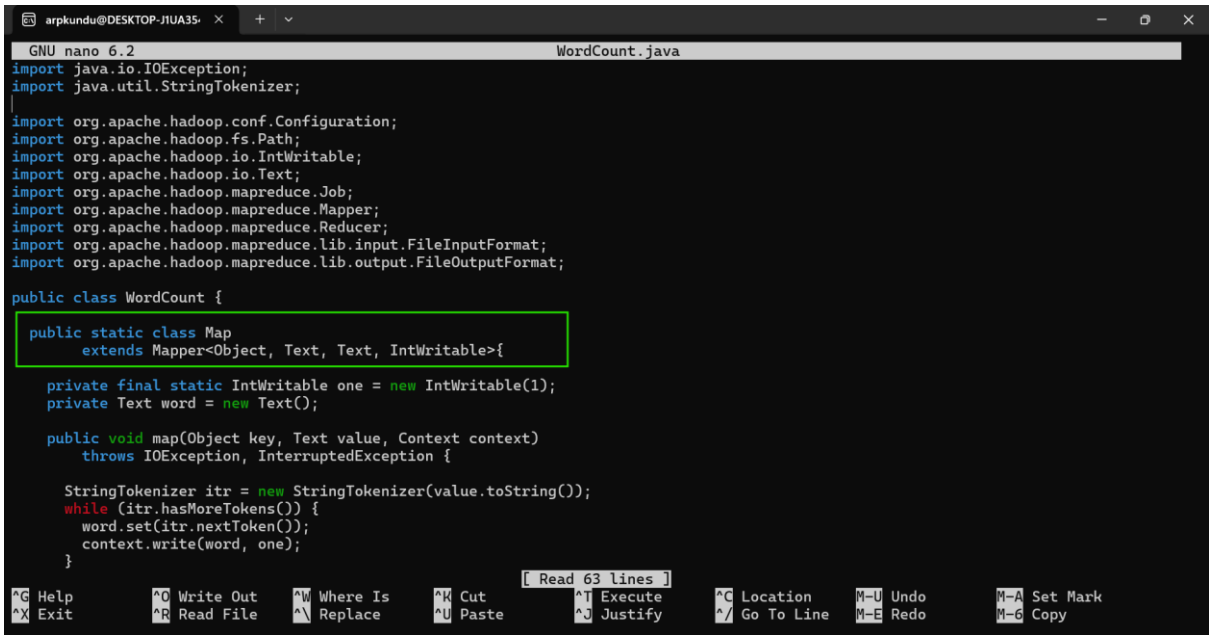
Q-4:

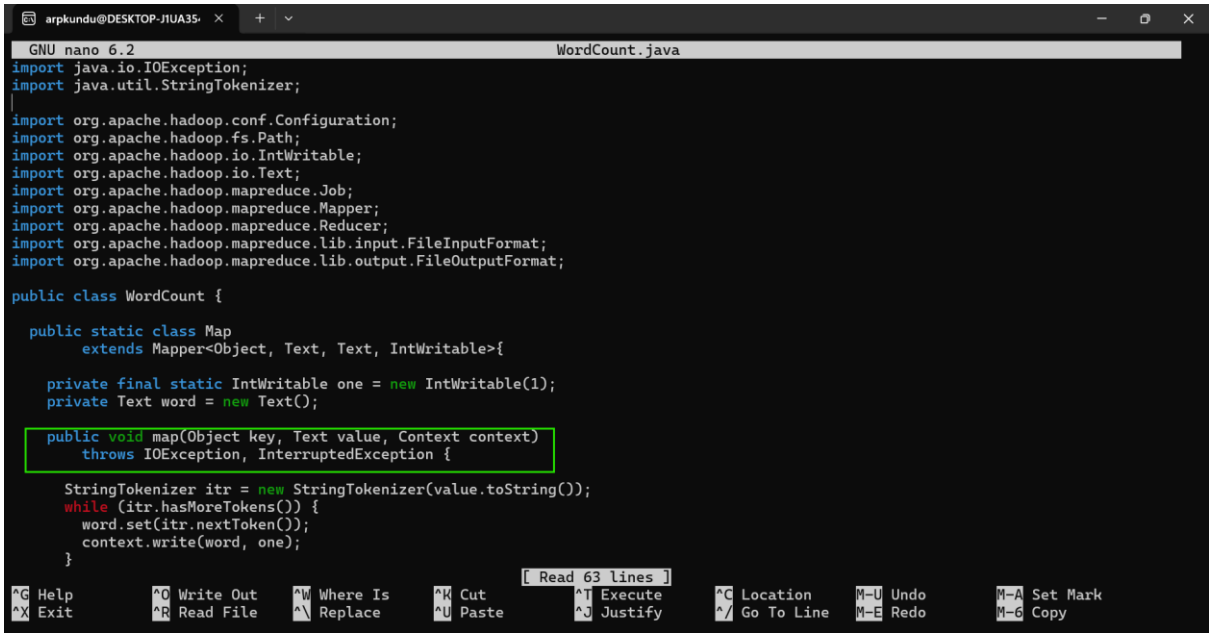
See the **WordCount.java** code from the MapReduce Tutorial, and copy its contents to the corresponding file in your project. Find the definitions of the Map class and the map() function:

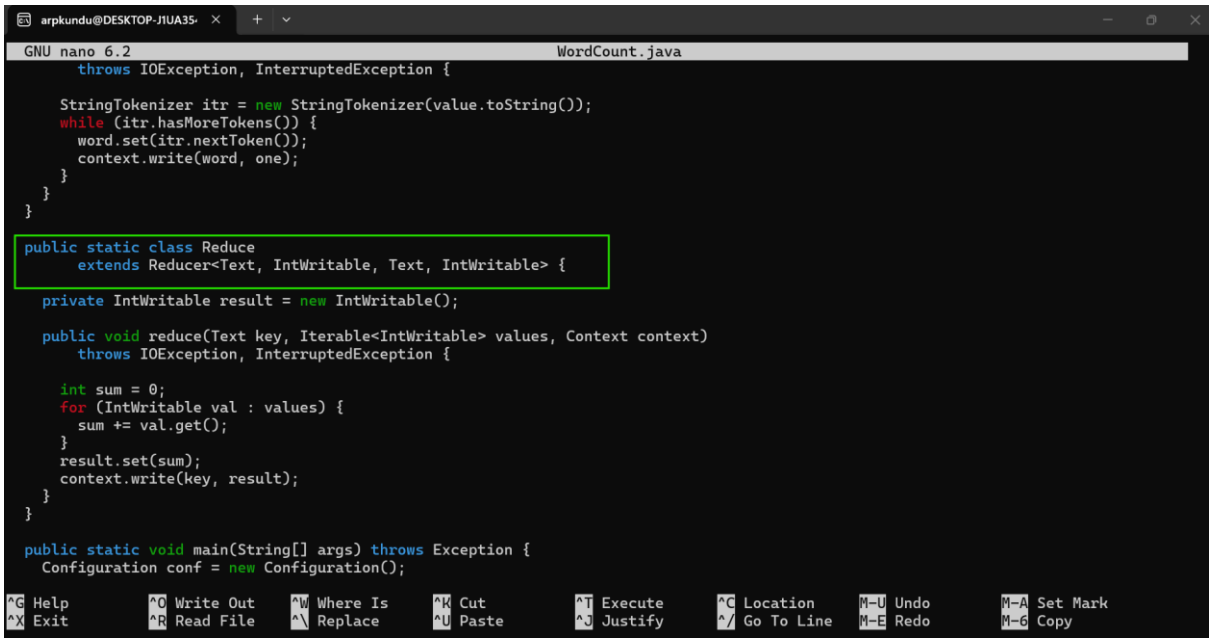
```
public static class Map extends Mapper {
@Override
public void map(/*?*/ key, /*?*/ value, Context context)
throws IOException, InterruptedException {
...
```

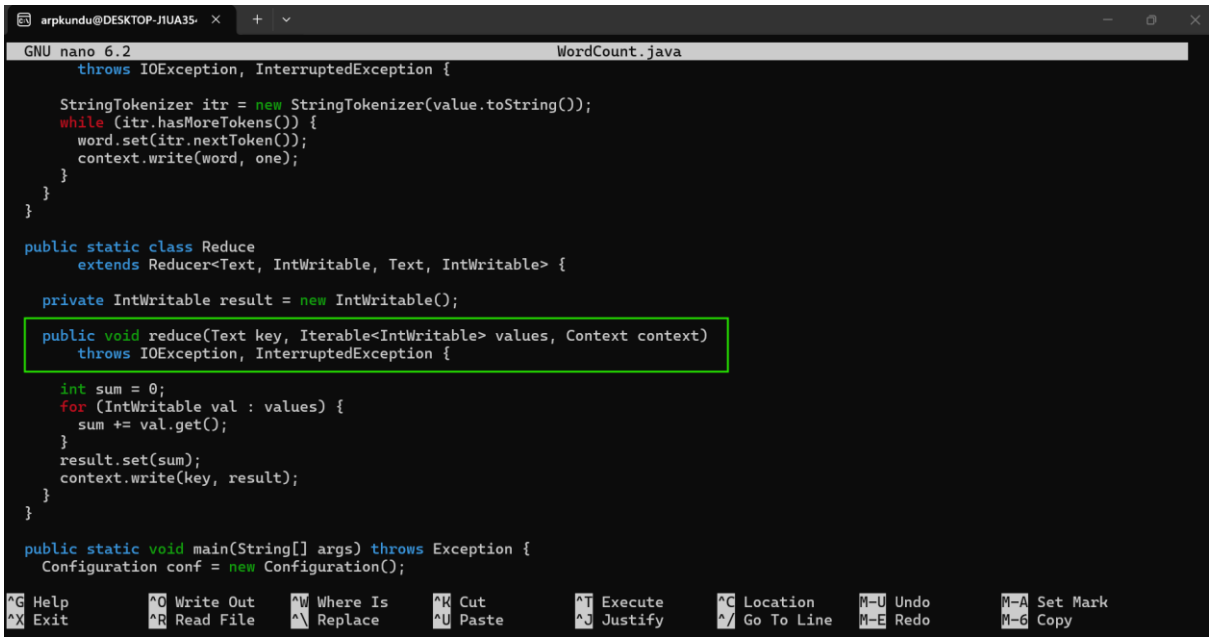
Use the data types you found in Question 1 to replace the /*?*/ placeholders.

Similarly, find the definitions of the Reduce class and the reduce() function and replace the /*?*/ placeholders with the data types found in Question 2.
You will also have to find which arguments to pass to job.setOutputKeyClass() and job.setOutputValueClass() .


arpkundu@DESKTOP-J1UA35
GNU nano 6.2 WordCount.java
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCount {
public static class Map
extends Mapper<Object, Text, Text, IntWritable>{
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
context.write(word, one);
}
[Read 63 lines]
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy


arpkundu@DESKTOP-J1UA35
GNU nano 6.2 WordCount.java
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
public class WordCount {
public static class Map
extends Mapper<Object, Text, Text, IntWritable>{
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
public void map(Object key, Text value, Context context)
throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
context.write(word, one);
}
[Read 63 lines]
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy


arpkundu@DESKTOP-J1UA35
GNU nano 6.2 WordCount.java
throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
context.write(word, one);
}
}
}
public static class Reduce
extends Reducer<Text, IntWritable, Text, IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy


arpkundu@DESKTOP-J1UA35
GNU nano 6.2 WordCount.java
throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
word.set(itr.nextToken());
context.write(word, one);
}
}
}
public static class Reduce
extends Reducer<Text, IntWritable, Text, IntWritable> {
private IntWritable result = new IntWritable();
public void reduce(Text key, Iterable<IntWritable> values, Context context)
throws IOException, InterruptedException {
int sum = 0;
for (IntWritable val : values) {
sum += val.get();
}
result.set(sum);
context.write(key, result);
}
}
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy

```
arpkundu@DESKTOP-J1UA35
 GNU nano 6.2                                                    WordCount.java
    private IntWritable result = new IntWritable();

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

      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Q-5:

Write the map() function. We want to make sure to disregard punctuation: to this end, you can use String.replaceAll() . In order to split lines into words, you can use a StringTokenizer .

```
public class WordCount {

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

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

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

            String line = value.toString()
                    .toLowerCase()
                    .replaceAll("[^a-z0-9\\s]", "");

            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
```

```
public class WordCount {

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

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

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

            String line = value.toString()
                    .toLowerCase()
                    .replaceAll("[^a-z0-9\\s]", "");

            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                context.write(word, one);
            }
        }
    }
```

Q-6:

Write the reduce() function. When you're done, make sure that compiling the project doesn't produce any errors.

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

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

          int sum = 0;
          for (IntWritable val : values) {
              sum += val.get();
          }
          context.write(key, new IntWritable(sum));
      }
  }
```

```
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "word count");

        job.setJarByClass(WordCount.class);
        job.setMapperClass(Map.class);
        job.setCombinerClass(Reduce.class);
        job.setReducerClass(Reduce.class);

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

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}
```

Run the Program without errors

```
arpkundu@DESKTOP-J1UA354:~$ ls
'WordCount$IntSumReducer.class'     mlbd
'WordCount$TokenizerMapper.class'   mlbd.pub
 WordCount.class                    wc.txt
 WordCount.java                     wc1.txt
 authorized_keys                    wordcount.jar
 authorized_keys.pub                wordcount_input.txt
 hadoop-3.3.6.tar.gz
arpkundu@DESKTOP-J1UA354:~$ javac -classpath `hadoop classpath` Word
Count.java
arpkundu@DESKTOP-J1UA354:~$ ls
'WordCount$IntSumReducer.class'     hadoop-3.3.6.tar.gz
'WordCount$Map.class'               mlbd
'WordCount$Reduce.class'            mlbd.pub
'WordCount$TokenizerMapper.class'   wc.txt
 WordCount.class                    wc1.txt
 WordCount.java                     wordcount.jar
 authorized_keys                    wordcount_input.txt
 authorized_keys.pub
```

```
arpkundu@DESKTOP-J1UA354:~$ hdfs dfs -cat /wc/output/part-r-00000
all     4
for     1
fun     1
get     2
good    1
lucky   1
night   4
some    1
sun     1
the     1
till    1
to      2
up      4
were    4
arpkundu@DESKTOP-J1UA354:~$
```

Q-7:

We will now run WordCount on the 200.txt file.

```
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$ ls
10.txt   13.txt   160.txt  206.txt  234.txt  263.txt  291.txt  318.txt  348.txt  375.txt  407.txt  44.txt   469.txt  496.txt  77.txt
101.txt  130.txt  161.txt  207.txt  235.txt  264.txt  292.txt  32.txt   349.txt  376.txt  408.txt  440.txt  47.txt   498.txt  78.txt
102.txt  131.txt  162.txt  208.txt  236.txt  265.txt  293.txt  321.txt  35.txt   377.txt  409.txt  442.txt  470.txt  499.txt  79.txt
103.txt  132.txt  163.txt  209.txt  237.txt  266.txt  295.txt  322.txt  350.txt  378.txt  41.txt   443.txt  471.txt  5.txt    8.txt
104.txt  133.txt  164.txt  21.txt   238.txt  267.txt  296.txt  323.txt  351.txt  379.txt  410.txt  444.txt  472.txt  50.txt   80.txt
105.txt  134.txt  165.txt  210.txt  239.txt  268.txt  297.txt  324.txt  352.txt  38.txt   411.txt  445.txt  473.txt  51.txt   81.txt
106.txt  136.txt  166.txt  211.txt  24.txt   27.txt   298.txt  326.txt  353.txt  380.txt  412.txt  446.txt  474.txt  52.txt   82.txt
107.txt  137.txt  167.txt  212.txt  240.txt  270.txt  299.txt  327.txt  354.txt  381.txt  414.txt  447.txt  475.txt  53.txt   83.txt
108.txt  138.txt  168.txt  213.txt  241.txt  271.txt  3.txt    328.txt  355.txt  382.txt  416.txt  448.txt  476.txt  54.txt   85.txt
109.txt  139.txt  169.txt  214.txt  242.txt  272.txt  30.txt   33.txt   356.txt  383.txt  417.txt  449.txt  477.txt  55.txt   87.txt
11.txt   14.txt   17.txt   215.txt  243.txt  273.txt  300.txt  330.txt  357.txt  384.txt  418.txt  45.txt   478.txt  56.txt   88.txt
110.txt  140.txt  170.txt  216.txt  244.txt  274.txt  301.txt  331.txt  358.txt  385.txt  419.txt  450.txt  479.txt  57.txt   9.txt
111.txt  143.txt  171.txt  217.txt  246.txt  275.txt  302.txt  332.txt  359.txt  389.txt  42.txt   451.txt  48.txt   58.txt   90.txt
112.txt  144.txt  172.txt  219.txt  249.txt  276.txt  303.txt  333.txt  360.txt  39.txt   420.txt  452.txt  480.txt  6.txt    92.txt
113.txt  145.txt  173.txt  22.txt   25.txt   277.txt  304.txt  334.txt  361.txt  390.txt  421.txt  453.txt  481.txt  60.txt   94.txt
114.txt  146.txt  174.txt  220.txt  250.txt  278.txt  305.txt  335.txt  362.txt  391.txt  422.txt  454.txt  482.txt  61.txt   96.txt
115.txt  148.txt  175.txt  221.txt  251.txt  279.txt  306.txt  336.txt  363.txt  392.txt  423.txt  455.txt  483.txt  62.txt   97.txt
117.txt  149.txt  176.txt  222.txt  252.txt  28.txt   307.txt  337.txt  364.txt  396.txt  424.txt  456.txt  484.txt  63.txt   98.txt
118.txt  15.txt   178.txt  223.txt  253.txt  280.txt  308.txt  338.txt  365.txt  397.txt  427.txt  457.txt  485.txt  64.txt   99.txt
12.txt   150.txt  179.txt  224.txt  254.txt  281.txt  309.txt  339.txt  366.txt  398.txt  428.txt  458.txt  486.txt  65.txt
121.txt  151.txt  18.txt   226.txt  255.txt  283.txt  31.txt   34.txt   367.txt  399.txt  429.txt  459.txt  487.txt  66.txt
122.txt  152.txt  180.txt  227.txt  256.txt  284.txt  310.txt  340.txt  368.txt  4.txt    430.txt  46.txt   488.txt  67.txt
123.txt  153.txt  19.txt   228.txt  257.txt  285.txt  311.txt  341.txt  369.txt  400.txt  431.txt  460.txt  489.txt  68.txt
124.txt  154.txt  2.txt    229.txt  258.txt  286.txt  312.txt  342.txt  37.txt   401.txt  432.txt  461.txt  49.txt   69.txt
125.txt  156.txt  20.txt   23.txt   259.txt  287.txt  313.txt  343.txt  370.txt  402.txt  433.txt  462.txt  490.txt  7.txt
126.txt  157.txt  200.txt  230.txt  26.txt   288.txt  314.txt  344.txt  371.txt  403.txt  434.txt  463.txt  491.txt  71.txt
127.txt  158.txt  201.txt  231.txt  260.txt  289.txt  315.txt  345.txt  372.txt  404.txt  436.txt  465.txt  493.txt  72.txt
128.txt  159.txt  202.txt  232.txt  261.txt  29.txt   316.txt  346.txt  373.txt  405.txt  437.txt  466.txt  494.txt  73.txt
129.txt  16.txt   204.txt  233.txt  262.txt  290.txt  317.txt  347.txt  374.txt  406.txt  439.txt  467.txt  495.txt  75.txt
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$ pwd
/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$
```

```
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$ pwd
/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$ ls | grep "^200.txt$"
200.txt
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$
```

Copy 200.txt into HDFS

```
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$ hdfs dfs -copyFromLocal 200.txt /user/arpkundu/
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$ hdfs dfs -ls /user/arpkundu
Found 1 items
-rw-r--r--   1 arpkundu supergroup    8312639 2026-02-10 19:22 /user/arpkundu/200.txt
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$
```

```
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$
hdfs dfs -ls output
-rw-r--r--   1     arpkundu supergroup   0 2026-02-10 19:25 output/_SUCCESS
-rw-r--r--   1     arpkundu supergroup 863649 2026-02-19:25 output/part-r-00000
arpkundu@DESKTOP-J1UA354:/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB$
Found 2 items
zurbriggen     1        zurbriggen 1
zurich        15        zurich 15
zurichgrebel   1        zurichgrebel 1
zurita         1        zurita 1
zurmat         1        zurmat 1
zusammenhang   1        zusammenhang 1
zutphen 1      1        zutphen 1
zweckmasigkeit 1        zweckmassigkeit 1
zwei 4         1        zwei 4
zweite 1                zweite 1
zweyer 1                zwolkau 2
zwickau 2               zwinglias 1
zwinglians 1            zygoma 1
zwinglius  1            zyryanovsk 1
zwolf   1               zygote 3
```

Q-8:

Why don't we have a replication factor for directories in HDFS?

In HDFS, **replication applies only to file blocks**, not directories.

1. **Directories Do Not Store Data Blocks:**
   - Files in HDFS are split into **blocks**.
   - Each block is replicated (e.g., 3 copies).
   - Directories **do not contain data blocks**.
   - They only store **metadata** (like file names and structure).

Since replication is about copying **data blocks across DataNodes**, directories don't need replication.

2. **Directory Information is Stored in the NameNode**

   The **NameNode** manages:
   - Directory structure
   - File names
   - Block locations

   This metadata is stored centrally (with FSImage and EditLogs).

   It is not distributed across DataNodes like file blocks.

So directories are already managed through metadata — not through block replication.

3. **Replicating Directories Would Be Redundant**

   If HDFS replicated directories:

   - It would duplicate only metadata references
   - But the actual data replication already happens at block level
   - So directory-level replication would serve no purpose

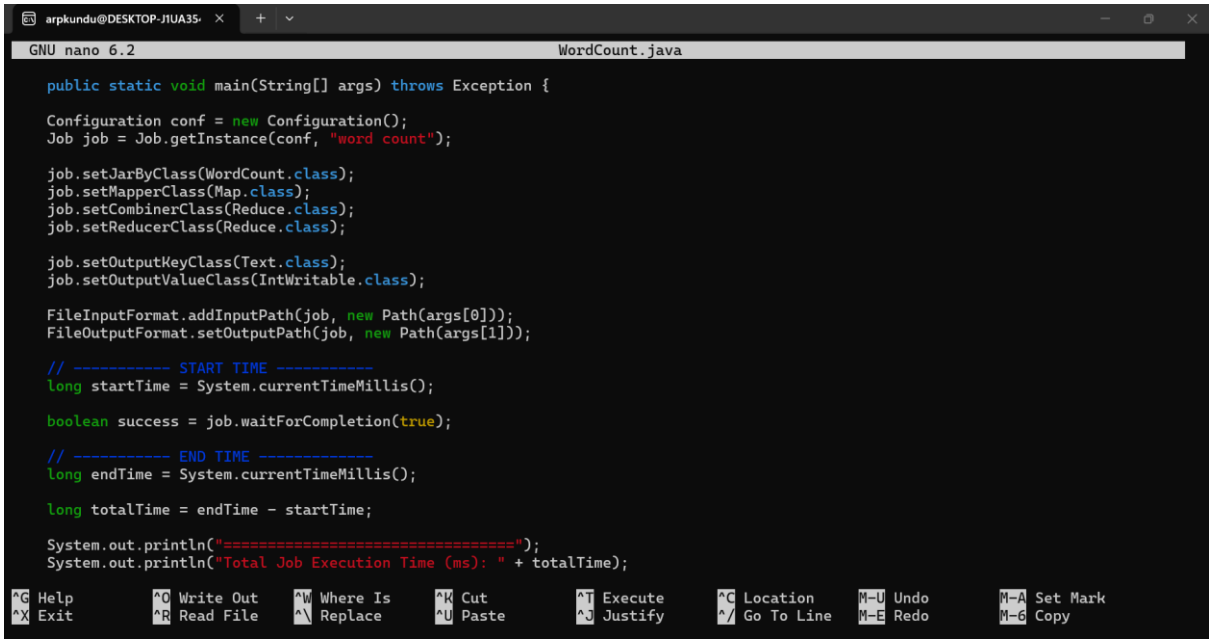
   Directories in HDFS do not have a replication factor because replication applies only to file blocks. Directories contain only metadata, which is managed by the NameNode. Since no data blocks are stored in directories, replication is unnecessary.

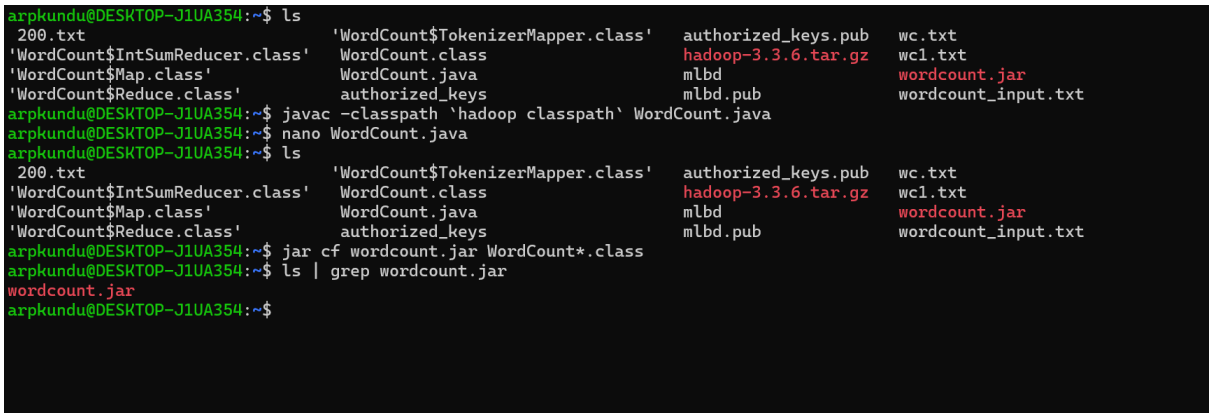
Q-9:

Edit WordCount.java to make it measure and display the total execution time of the job. Experiment with the mapreduce.input.fileinputformat.split.maxsize parameter. You can change its value using:

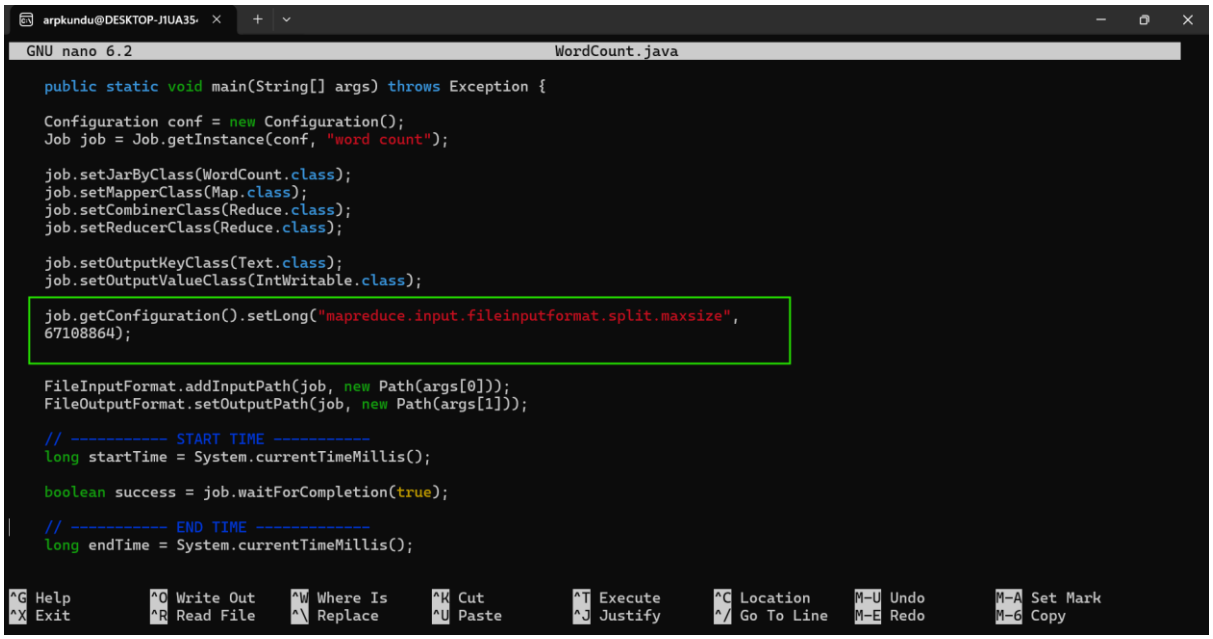
```
job.getConfiguration().setLong("mapreduce.input.fileinputformat.split.maxsize",);
```

PART 1 : Measure and display total execution time





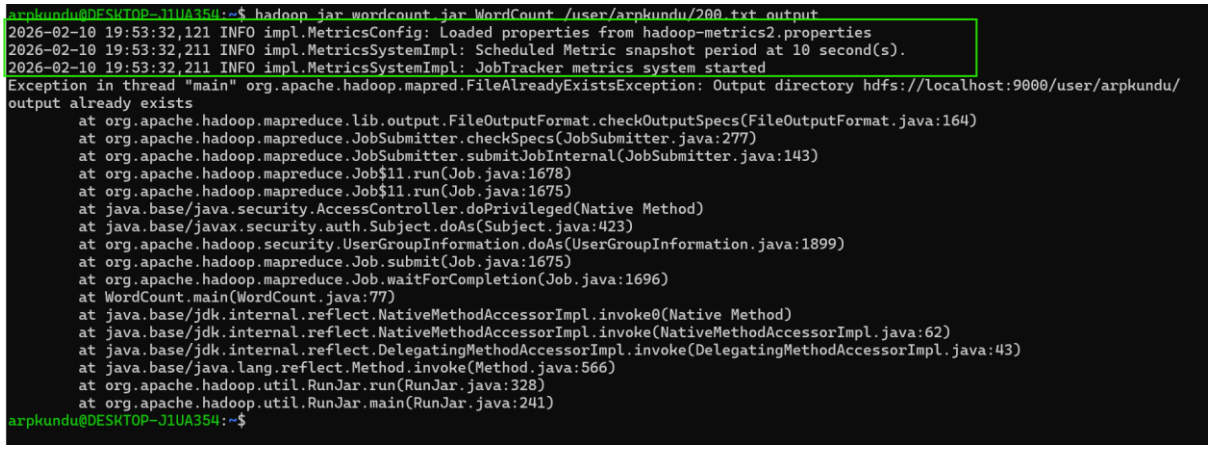
PART 2: Experiment with split.maxsize



Compile : javac -classpath `hadoop classpath` WordCount.java

Create jar:
jar cf wordcount.jar WordCount*.class

Run the job:



How does changing mapreduce.input.fileinputformat.split.maxsize impact performance? Why?

mapreduce.input.fileinputformat.split.maxsize

- controls the **maximum size of input splits**, which determines how many **Mapper tasks** are created.

What Happens When You Change It?

Smaller Split Size (e.g., 32MB)

- Creates more input splits
- More Map tasks

If too small → performance may decrease due to overhead.

Larger Split Size (e.g., 256MB or 512MB)

- Creates fewer input splits
- Fewer Map tasks

If too large → performance may decrease due to underutilization.

Why This Happens?

Hadoop processes data **per split**, and:

- 1 split = 1 Mapper
- More splits = more parallel processing
- But too many small tasks = scheduling + context switching overhead
- Too few large tasks = not enough parallelism

So performance depends on **balancing parallelism vs overhead**.

Ex-

Suppose:

- File size = 1GB
- Default block size = 128MB

Default:

1GB / 128MB ≈ 8 Mappers

If split.maxsize = 32MB:

1GB / 32MB ≈ 32 Mappers
More parallel but more overhead

If split.maxsize = 512MB:

1GB / 512MB ≈ 2 Mappers
Less overhead but low parallelism

Usually:

- Keep split size close to **HDFS block size**
- Tune based on:
  - Cluster size
  - CPU cores
  - Dataset size

Changing mapreduce.input.fileinputformat.split.maxsize affects the number of input splits and therefore the number of mapper tasks. Smaller split sizes increase parallelism but add scheduling overhead, while larger split sizes reduce overhead but may underutilize cluster resources. Optimal performance is achieved by balancing parallelism and task management overhead, usually by keeping split size close to the HDFS block size.

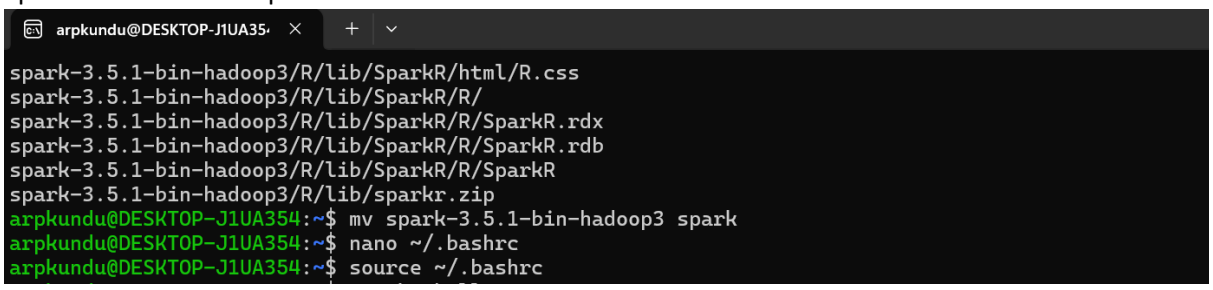
Apache Spark

Note: For these Spark problems, first load the provided Project Gutenberg dataset into a Spark DataFrame named books_df with the schema:

- **file_name (string): The name of the text file (e.g., "10.txt").**
- **text (string): The raw text content of the book.**

Q-10:

Spark installation completed.



```
arpkundu@DESKTOP-J1UA354:~$ pyspark
Python 3.10.12 (main, Jan 26 2026, 14:55:28) [GCC 11.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
26/02/10 20:29:26 WARN Utils: Your hostname, DESKTOP-J1UA354 resolves to a loopback address: 127
(on interface lo)
26/02/10 20:29:26 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
26/02/10 20:29:27 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform
pplicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.1
      /_/

Using Python version 3.10.12 (main, Jan 26 2026 14:55:28)
Spark context Web UI available at http://10.255.255.254:4040
Spark context available as 'sc' (master = local[*], app id = local-1770735568076).
SparkSession available as 'spark'.
>>>
```

Book Metadata Extraction and Analysis:

PART 1: Metadata Extraction

```
>>> from pyspark.sql import Row
>>> from pyspark.sql.functions import regexp_extract
>>> rdd = spark.sparkContext.wholeTextFiles(
nt/c/Users/ARPIT...     "/mnt/c/Users/ARPITA KUNDU/Downloads/D184MB/D184MB/*.txt"
... )
>>> books_df = rdd.map(lambda x: Row(
...     file_name=x[0].split("/")[-1],
...     text=x[1]
... )).toDF()
```

```
+---------+--------------------------------------------------+--------------------+--------+--------+
|file_name|title                                             |release_date        |language|encoding|
+---------+--------------------------------------------------+--------------------+--------+--------+
|10.txt   |The King James Bible                              |March 2, 2011 [EBook #10]|English |ASCII   |
|101.txt  |Hacker Crackdown                                  |January, 1994       |English |ASCII   |
|102.txt  |The Tragedy of Pudd'nhead Wilson                  |January, 1994       |English |ASCII   |
|103.txt  |Around the World in 80 Days                       |May 15, 2008 [EBook #103]|English |ASCII   |
|104.txt  |Franklin Delano Roosevelt's First Inaugural Address|May 14, 2008 [EBook #104]|English |ASCII   |
+---------+--------------------------------------------------+--------------------+--------+--------+
only showing top 5 rows
```

PART 2: Analysis

Number of books released each year:

```
>>> metadata_df = metadata_df.withColumn(
...     "release_year",
...     regexp_extract(col("release_date"), r"(\d{4})", 1)
... )
>>>
```

```
>>> books_per_year = metadata_df \
upBy("release_year") \
    .count() \
    .orderBy("release_year")

books_per_year.show()
...     .filter(col("release_year") != "") \
...     .groupBy("release_year") \
...     .count() \
...     .orderBy("release_year")
>>>
```

```
>>> books_per_year.show()
+------------+-----+
|release_year|count|
+------------+-----+
|        1975|    1|
|        1978|    1|
|        1979|    1|
|        1991|    7|
|        1992|   19|
|        1993|   13|
|        1994|   17|
|        1995|   60|
|        1996|   53|
|        2002|    1|
|        2004|    7|
|        2005|    4|
|        2006|   42|
|        2007|   13|
|        2008|  154|
|        2009|    1|
|        2010|    9|
|        2011|    1|
|        2012|    2|
|        2013|    1|
+------------+-----+
only showing top 20 rows
```

Most common language in the dataset:

```
>>> metadata_df \
...     .filter(col("language") != "") \
...     .groupBy("language") \
...     .count() \
...     .orderBy(col("count").desc()) \
...     .show(1)
+--------+-----+
|language|count|
+--------+-----+
| English|  404|
+--------+-----+
only showing top 1 row

>>>
```

Average length of book titles (characters):

```
>>> metadata_df \
...     .filter(col("title") != "") \
...     .select(length(col("title")).alias("title_length")) \
...     .agg({"title_length": "avg"}) \
...     .show()
+------------------+
| avg(title_length)|
+------------------+
|22.829268292682926|
+------------------+
```

PART 3:

Explain the regular expressions you used to extract the title, release date, language, and encoding. Discuss any challenges or limitations in using regular expressions for this task.

Regular expressions used in task:

Title:

(?i)Title:\s*(.+) :
(?i) → case-insensitive
Matches the title line and captures the full title text.

Release Date:

(?i)Release Date:\s*(.+)
Captures the release date line
Year extracted later using \d{4}

Language:

(?i)Language:\s*(.+)
Extracts language metadata (ex- English)

Encoding:

(?i)Character set encoding:\s*(.+)
Extracts text encoding (e.g., UTF-8)

Challenges / Limitations

Some major challenges:

1. **Missing metadata** : Some books do not contain all fields
2. **Inconsistent formatting** : Metadata labels may vary across books
3. **Multiple matches** : Regex captures only the first occurrence
4. **Noise in text** : Metadata may appear outside the header

PART 4:

What are some potential issues with the extracted metadata (e.g., inconsistencies, missing values)? How would you handle these issues in a real-world scenario?

How to handle in real-world scenarios?

1. Filter empty values.
2. Normalize text before extraction.
3. Use structured metadata formats (XML / JSON).
4. Combine regex with NLP-based extraction.
5. Apply data validation and defaults.

- Metadata such as title, release date, language, and encoding were extracted from the raw text using regular expressions applied to each book loaded as a single record.
- Apache Spark's regexp_extract function was used for efficient metadata extraction. The number of books released per year was calculated by extracting the year from the release date. The most common language was identified using frequency analysis, and the average title length was computed using string length aggregation.
- Regular expressions may fail when metadata is missing or inconsistently formatted; in real-world applications, such issues can be addressed using normalization, validation, and structured metadata sources.

Q-11:

TF-IDF and Book Similarity:

Task: Calculate TF-IDF scores for words in each book and use them to determine book similarity based on cosine similarity.

PART 1: Preprocessing

Clean the text column in books_df : remove Project Gutenberg header/footer, convert to lowercase, remove punctuation, tokenize into words, and remove stop words.

Data Cleaning:

Remove:

- Gutenberg header/footer (simple heuristic)
- Convert to lowercase
- Remove punctuation
- Tokenize
- Remove stopwords

Import required libraries:

```
Using Python version 3.10.12 (main, Jan 26 2026 14:55:28)
Spark context Web UI available at http://10.255.255.254:4040
Spark context available as 'sc' (master = local[*], app id = local-1770803791182).
SparkSession available as 'spark'.
>>> from pyspark.ml.feature import Tokenizer, StopWordsRemover
>>> from pyspark.sql.functions import col, lower, regexp_replace
>>>
```

```
+---------+--------------------------------------------------+--------------------------+--------+--------------------+
|file_name|title                                             |release_date              |language|encoding            |
+---------+--------------------------------------------------+--------------------------+--------+--------------------+
|10.txt   |The King James Bible                              |March 2, 2011 [EBook #10] |English |ASCII               |
|101.txt  |Hacker Crackdown                                  |January, 1994             |English |ASCII               |
|102.txt  |The Tragedy of Pudd'nhead Wilson                  |January, 1994             |English |ASCII               |
|103.txt  |Around the World in 80 Days                       |May 15, 2008 [EBook #103] |English |ASCII               |
|104.txt  |Franklin Delano Roosevelt's First Inaugural Address|May 14, 2008 [EBook #104] |English |ASCII               |
|105.txt  |Persuasion                                        |June 5, 2008 [EBook #105] |English |ASCII               |
|106.txt  |Jungle Tales of Tarzan                            |June 5, 2008 [EBook #106] |English |ASCII               |
|107.txt  |Far from the Madding Crowd                        |February, 1994  [eBook #107]|English |ISO-646-US (US-ASCII)|
|108.txt  |The Return of Sherlock Holmes                     |July 8, 2007 [EBook #108] |English |ASCII               |
|109.txt  |Renascence and Other Poems                        |June 19, 2008 [EBook #109]|English |ASCII               |
+---------+--------------------------------------------------+--------------------------+--------+--------------------+
only showing top 10 rows
```

Data Cleaning: convert to lowercase and remove punctuation

```
>>> clean_df = books_df.withColumn(
    "clean_text",
    lower(rege...     "clean_text",
...     lower(regexp_replace(col("text"), "[^a-zA-Z\\s]", " "))
... )
```

```
>>> clean_df.show()
26/02/11 15:42:13 WARN PythonRunner: Detected deadlock while completing task 0.0
+---------+--------------------+--------------------+
|file_name|                text|          clean_text|
+---------+--------------------+--------------------+
|   10.txt|The Project Guten...|the project guten...|
|  101.txt|The Project Guten...|the project guten...|
|  102.txt|The Project Guten...|the project guten...|
|  103.txt|The Project Guten...|the project guten...|
|  104.txt|The Project Guten...|the project guten...|
|  105.txt|The Project Guten...|the project guten...|
|  106.txt|Project Gutenberg...|project gutenberg...|
|  107.txt|The Project Guten...|the project guten...|
|  108.txt|Project Gutenberg...|project gutenberg...|
|  109.txt|Project Gutenberg...|project gutenberg...|
|   11.txt|Project Gutenberg...|project gutenberg...|
|  110.txt|The Project Guten...|the project guten...|
|  111.txt|The Project Guten...|the project guten...|
|  112.txt|The Project Guten...|the project guten...|
|  113.txt|The Project Guten...|the project guten...|
|  114.txt|The Project Guten...|the project guten...|
|  115.txt|The Project Guten...|the project guten...|
|  117.txt|The Project Guten...|the project guten...|
|  118.txt|The Project Guten...|the project guten...|
|   12.txt|The Project Guten...|the project guten...|
+---------+--------------------+--------------------+
only showing top 20 rows
```

Tokenize :

```
>>> tokenizer = Tokenizer(inputCol="clean_text", outputCol="words")
rm(clean_df)
>>> tokenized_df = tokenizer.transform(clean_df)
>>>
```

Remove stop words:

```
>>> remover = StopWordsRemover(inputCol="words", outputCol="filtered_words")
ransform(tokenized_df)
>>> filtered_df = remover.transform(tokenized_df)
>>>
```

PART 2: TF-IDF Calculation

1. Calculate the Term Frequency (TF) of each word in each book.
   Import library

```
>>> from pyspark.ml.feature import HashingTF, IDF
>>>
```

Term Frequency (TF):

```
>>> hashingTF = HashingTF(
Col="raw_feature...     inputCol="filtered_words",
s",
    numFeatu...     outputCol="raw_features",
res=10000
)

fea...     numFeatures=10000
hingTF.transform... )
(filtered_df)
>>>
>>> featurized_df = hashingTF.transform(filtered_df)
>>>
```

Inverse Document Frequency (IDF):

2. Calculate the Inverse Document Frequency (IDF) for each word across all books.
3. Compute the TF-IDF score for each word in each book (TF * IDF).

PART 3: Book Similarity

1. Represent each book as a vector of its TF-IDF scores.
2. Calculate the cosine similarity between all pairs of book vectors.
3. For a given book (e.g., file_name "10.txt"), identify the top 5 most similar books based on cosine similarity.

PART 4:

Explain TF and IDF. Why is TF-IDF useful for weighting the importance of words in a document?

What is TF?

- Term Frequency (TF) measures how often a word appears in a document.

$$TF = \frac{\text{word count in document}}{\text{total words in document}}$$

What is IDF?

- Inverse Document Frequency measures how rare a word is across documents.

$$IDF = \log\left(\frac{N}{df}\right)$$

- N = total number of documents
- df = number of documents containing the word

Why is TF-IDF useful for weighting?

- Increases weight of important words and Suppresses common words (e.g., “love”, “the”, “and”)
- Reduces weight of common words
- Highlights distinguishing terms
- Improves document comparison and helps identify document themes
- Improves similarity detection

Suppose we have two books:

Book A (Romance) : love love love marriage happiness love
Book B (Science) : quantum physics atom energy quantum experiment

Step 1: Term Frequency (TF)

In Book A:

- **love** appears 4 times → High TF
- **marriage** appears 1 time

In Book B:

- **quantum** appears 2 times
- **physics** appears 1 time

So TF measures frequency **within the document only**.

Step 2: Inverse Document Frequency (IDF)

Now imagine we have **1,000 books** in the dataset.

- The word **“love”** appears in 800 books → Very common
- The word **“quantum”** appears in only 10 books → Rare

IDF will:

- Give **low weight** to "love"
- Give **high weight** to "quantum"

Final TF-IDF Effect :

Word	TF	IDF	TF-IDF Result
love	High	Low	Medium
quantum	Medium	High	High

Even though “love” appears many times,
it gets **lower importance** because it appears in many books.

But “quantum” gets a **higher score** because:

- It is rare across books
- It helps distinguish science books from romance books

In short :

For example, the word “love” may appear frequently in many books, so its IDF is low and its TF-IDF score is reduced. However, a rare word like “quantum” appears in only a few books, giving it a high IDF and higher TF-IDF score. This makes TF-IDF effective for identifying distinguishing words in documents.

PART 5:

How is cosine similarity calculated? Why is it appropriate for comparing documents represented as TF-IDF vectors?

How is Cosine Similarity Calculated?

Cosine similarity measures the **angle between two vectors**.

For two TF-IDF vectors **A** and **B**:

$$\text{cosine similarity} = \frac{A \cdot B}{||A||\,||B||}$$

Where:

- A·B = dot product of the two vectors
- ||A|| = magnitude (length) of vector A
- ||B|| = magnitude of vector B

Interpretation:

Value		Meaning
1	-	Documents are very similar
0	-	Documents are very similar
-1	-	Documents are unrelated

Since TF-IDF values are non-negative, cosine similarity is usually between **0 and 1**.

Why Cosine Similarity is Appropriate for TF-IDF?

1. Why Cosine Similarity is Appropriate for TF-IDF:
    - Longer documents naturally have larger vector magnitudes.
    - Cosine similarity **normalizes by vector length**, so length doesn’t bias similarity.

2. Independent of document length:

    TF-IDF vectors are:

- High dimensional
- Mostly zeros

Cosine similarity handles sparse vectors efficiently.

3. Works well with sparse vectors:
    - It measures **direction**, not magnitude.
    - So it compares topic similarity rather than document length.

In short:

Cosine similarity is calculated as the dot product of two vectors divided by the product of their magnitudes. It measures the angle between TF-IDF vectors. It is appropriate because it normalizes document length, works well with sparse high-dimensional vectors, and captures similarity based on word distribution rather than document size.

PART 6:

Discuss scalability challenges when calculating pairwise cosine similarity for many documents. How could Spark help address these challenges?

Scalability Challenges in Pairwise Cosine Similarity:

If there are **N documents**, computing all pairwise similarities requires: O(N^2) comparisons.

Example:

- 1,000 books → 1,000,000 comparisons
- 10,000 books → 100,000,000 comparisons
- 100,000 books → 10 billion comparisons

This becomes:

- Computationally expensive
- Memory intensive
- Slow for large datasets

How Spark Helps:

- Distributed Processing - Spark distributes similarity computation across multiple nodes.
- Parallel Vector Operations - TF-IDF vectors are processed in parallel using Spark MLlib.
- Efficient Data Partitioning - Spark partitions data across executors, enabling parallel comparisons.
- Approximate Methods –

    Spark provides:

    - **LSH (Locality Sensitive Hashing)**
    - **Approximate nearest neighbors**

    These reduce computation from $O(N^2)$ to much faster approximate methods.

In short:

Cosine similarity is calculated as the dot product of two TF-IDF vectors divided by the product of their magnitudes. It measures the angle between vectors and ranges from 0 to 1 for TF-IDF representations. It is appropriate for document comparison because it normalizes document length, handles sparse high-dimensional vectors effectively, and captures similarity based on word distribution rather than size. However, computing pairwise cosine similarity for N documents requires $O(N^2)$ comparisons, which becomes computationally expensive for large datasets. Apache Spark addresses this challenge through distributed processing, parallel computation, and approximate similarity techniques such as Locality Sensitive Hashing.

Q-12:

Author Influence Network:

Task:
Construct a simple author influence network based on whether authors' books were released within a certain time window of each other. This is a simplified representation, as true influence is much more complex, but it demonstrates graph-like analysis with Spark.

PART 1- Preprocessing:

STEP 1: Extract Author and Release Year

Extract the author and release_date from the text of each book, similar to what you did in the first question of the previous set. You might need to refine your regular expressions or use more advanced techniques if the format varies significantly.

```
+---------+-----------------------------------------------+--------------------------+--------+--------------------+
|file_name|title                                          |release_date              |language|encoding            |
+---------+-----------------------------------------------+--------------------------+--------+--------------------+
|10.txt   |The King James Bible                           |March 2, 2011 [EBook #10] |English |ASCII               |
|101.txt  |Hacker Crackdown                               |January, 1994             |English |ASCII               |
|102.txt  |The Tragedy of Pudd'nhead Wilson               |January, 1994             |English |ASCII               |
|103.txt  |Around the World in 80 Days                    |May 15, 2008 [EBook #103] |English |ASCII               |
|104.txt  |Franklin Delano Roosevelt's First Inaugural Address|May 14, 2008 [EBook #104]|English |ASCII               |
|105.txt  |Persuasion                                     |June 5, 2008 [EBook #105] |English |ASCII               |
|106.txt  |Jungle Tales of Tarzan                         |June 5, 2008 [EBook #106] |English |ASCII               |
|107.txt  |Far from the Madding Crowd                     |February, 1994  [eBook #107]|English |ISO-646-US (US-ASCII)|
|108.txt  |The Return of Sherlock Holmes                  |July 8, 2007 [EBook #108] |English |ASCII               |
|109.txt  |Renascence and Other Poems                     |June 19, 2008 [EBook #109]|English |ASCII               |
+---------+-----------------------------------------------+--------------------------+--------+--------------------+
only showing top 10 rows
```

```
>>> from pyspark.sql.functions import regexp_extract, col
>>> author_df = books_df \

        "release_date",
       ...      .withColumn(
 regexp_extract(...          "author",
col("text"), r"(...          regexp_extract(col("text"), r"(?i)Author:\s*(.+)", 1)
?i)Release Date:...      ) \
...      .withColumn(
...          "release_date",
...          regexp_extract(col("text"), r"(?i)Release Date:\s*(.+)", 1)
...      )
```

```
>>> author_df.show()
26/02/11 18:35:58 WARN PythonRunner: Detected deadlock while completing task 0.0 in stage 3 (TID 428): Attempting to kill Python Work
er
+---------+--------------------+--------------------+--------------------+
|file_name|                text|              author|        release_date|
+---------+--------------------+--------------------+--------------------+
|   10.txt|The Project Guten...|                    |March 2, 2011 [EB...|
|  101.txt|The Project Guten...|      Bruce Sterling|       January, 1994|
|  102.txt|The Project Guten...|Mark Twain (Samue...|       January, 1994|
|  103.txt|The Project Guten...|         Jules Verne|May 15, 2008 [EBo...|
|  104.txt|The Project Guten...|Franklin Delano R...|May 14, 2008 [EBo...|
|  105.txt|The Project Guten...|         Jane Austen|June 5, 2008 [EBo...|
|  106.txt|Project Gutenberg...|Edgar Rice Burroughs|June 5, 2008 [EBo...|
|  107.txt|The Project Guten...|        Thomas Hardy|February, 1994  [...|
|  108.txt|Project Gutenberg...|  Arthur Conan Doyle|July 8, 2007 [EBo...|
|  109.txt|Project Gutenberg...|Edna St. Vincent ...|June 19, 2008 [EB...|
|   11.txt|Project Gutenberg...|       Lewis Carroll|         March, 1994|
|  110.txt|The Project Guten...|        Thomas Hardy|February, 1994  [...|
|  111.txt|The Project Guten...|Gene Stratton-Porter|March 8, 2006 [EB...|
|  112.txt|The Project Guten...|     Richard McGowan|         March, 1994|
|  113.txt|The Project Guten...|Frances Hodgson B...|May 15, 2008 [EBo...|
|  114.txt|The Project Guten...|    Sir John Tenniel|May 27, 2008 [EBo...|
|  115.txt|The Project Guten...|U.S. Census of Po...|June 5, 2008 [EBo...|
|  117.txt|The Project Guten...|Ludwig van Beethoven|March 16, 2012 [E...|
|  118.txt|The Project Guten...|Electronic Fronti...|         March, 1994|
|   12.txt|The Project Guten...|Charles Dodgson, ...|      February, 1991|
+---------+--------------------+--------------------+--------------------+
only showing top 20 rows
```

STEP 2: Extract Release Year

```
>>> author_df = author_df.withColumn(
...     "release_year",
...     regexp_extract(col("release_date"), r"(\d{4})", 1)
... )
>>> author_df.show
```

```
>>> author_df.show()
26/02/11 18:38:34 WARN PythonRunner: Detected deadlock while completing task 0.0 in stage 4 (TID
er
+---------+--------------------+--------------------+--------------------+------------+
|file_name|                text|              author|        release_date|release_year|
+---------+--------------------+--------------------+--------------------+------------+
|   10.txt|The Project Guten...|                    |March 2, 2011 [EB...|        2011|
|  101.txt|The Project Guten...|      Bruce Sterling|       January, 1994|        1994|
|  102.txt|The Project Guten...|Mark Twain (Samue...|       January, 1994|        1994|
|  103.txt|The Project Guten...|         Jules Verne|May 15, 2008 [EBo...|        2008|
|  104.txt|The Project Guten...|Franklin Delano R...|May 14, 2008 [EBo...|        2008|
|  105.txt|The Project Guten...|         Jane Austen|June 5, 2008 [EBo...|        2008|
|  106.txt|Project Gutenberg...|Edgar Rice Burroughs|June 5, 2008 [EBo...|        2008|
|  107.txt|The Project Guten...|        Thomas Hardy|February, 1994  [...|        1994|
|  108.txt|Project Gutenberg...|  Arthur Conan Doyle|July 8, 2007 [EBo...|        2007|
|  109.txt|Project Gutenberg...|Edna St. Vincent ...|June 19, 2008 [EB...|        2008|
|   11.txt|Project Gutenberg...|       Lewis Carroll|         March, 1994|        1994|
|  110.txt|The Project Guten...|        Thomas Hardy|February, 1994  [...|        1994|
|  111.txt|The Project Guten...|Gene Stratton-Porter|March 8, 2006 [EB...|        2006|
|  112.txt|The Project Guten...|     Richard McGowan|         March, 1994|        1994|
|  113.txt|The Project Guten...|Frances Hodgson B...|May 15, 2008 [EBo...|        2008|
|  114.txt|The Project Guten...|    Sir John Tenniel|May 27, 2008 [EBo...|        2008|
|  115.txt|The Project Guten...|U.S. Census of Po...|June 5, 2008 [EBo...|        2008|
|  117.txt|The Project Guten...|Ludwig van Beethoven|March 16, 2012 [E...|        2012|
|  118.txt|The Project Guten...|Electronic Fronti...|         March, 1994|        1994|
|   12.txt|The Project Guten...|Charles Dodgson, ...|      February, 1991|        1991|
+---------+--------------------+--------------------+--------------------+------------+
only showing top 20 rows
```

```
>>> author_year_df = author_df.select("author", "release_year")
>>> author_year_df = author_year_df.dropDuplicates()
>>> author_year_df = author_year_df.orderBy("release_year")
>>> from pyspark.sql.functions import col
") \
    .filter(
        (col("author").isNotNull()) &
        (col("release_year").isNotNull()>>>
>>> author_year_df = author_df.select("author", "release_year") \
...     .filter(
...         (col("author").isNotNull()) &
...         (col("release_year").isNotNull())
...     )
>>>
```

```
>>> author_year_df.show()
26/02/11 18:44:44 WARN PythonRunner: Detected
er
+--------------------+------------+
|              author|release_year|
+--------------------+------------+
|      Bruce Sterling|        1994|
|Mark Twain (Samue...|        1994|
|         Jules Verne|        2008|
|Franklin Delano R...|        2008|
|         Jane Austen|        2008|
|Edgar Rice Burroughs|        2008|
|        Thomas Hardy|        1994|
|  Arthur Conan Doyle|        2007|
|Edna St. Vincent ...|        2008|
|       Lewis Carroll|        1994|
|        Thomas Hardy|        1994|
|Gene Stratton-Porter|        2006|
|     Richard McGowan|        1994|
|Frances Hodgson B...|        2008|
|    Sir John Tenniel|        2008|
|U.S. Census of Po...|        2008|
|Ludwig van Beethoven|        2012|
|Electronic Fronti...|        1994|
|Charles Dodgson, ...|        1991|
|         Jane Austen|        1994|
+--------------------+------------+
only showing top 20 rows
```

PART 2: Construct Influence Network

1. Define an "influence" relationship between two authors if one author released a book within X years of another author's release (e.g., X = 5 years). You can adjust X to explore different influence ranges.

2. Create an RDD or DataFrame representing the edges of this influence network. Each edge should be a tuple or row of the form (author1, author2), indicating that author1 potentially influenced author2 .

Author A influences Author B if: $0<(yearB-yearA)\leq X$

```
>>> from pyspark.sql.functions import abs
>>> a = author_df.alias("a")
>>> b = author_df.alias("b")
```

```
>>> X = 5
```

```
>>> edges_df = a.join(b, col("a.author") != col("b.author")) \
year").cast("int") > col("a.release_year").cast(...      .filter(
...         (col("b.release_year").cast("int") > col("a.release_year").cast("int")) &
se_year").cast("...         ((col("b.release_year").cast("int") - col("a.release_year").cast("int")) <= X)
lect(
       co...     ) \
...     .select(
...         col("a.author").alias("author1"),
...         col("b.author").alias("author2")
...     )
```

```
>>> edges_df.show()
26/02/11 18:52:29 WARN PythonRunner: Det
er
+--------------+--------------------+
|       author1|             author2|
+--------------+--------------------+
|Bruce Sterling|   Project Gutenberg|
|Bruce Sterling|      Robert Service|
|Bruce Sterling|         Henry James|
|Bruce Sterling|             Unknown|
|Bruce Sterling|Andrew Barton 'Ba...|
|Bruce Sterling|             Lao-Tze|
|Bruce Sterling|       Joseph Conrad|
|Bruce Sterling|       Joseph Conrad|
|Bruce Sterling|    G. K. Chesterton|
|Bruce Sterling|    Theodore Dreiser|
|Bruce Sterling|      Giuseppe Salza|
|Bruce Sterling|       United States|
|Bruce Sterling|       Robert Harris|
|Bruce Sterling|  Arthur Conan Doyle|
|Bruce Sterling|        Omar Khayyam|
|Bruce Sterling|       Jean Clottes|
|Bruce Sterling|       Michael Hart|
|Bruce Sterling|       United States|
|Bruce Sterling|        Dan Scavelle|
|Bruce Sterling|       United States|
+--------------+--------------------+
only showing top 20 rows
```

PART 3: Compute In-Degree and Out-Degree

1. Calculate the "in-degree" and "out-degree" of each author in the network. In-degree represents the number of authors who potentially influenced them, and out-degree represents the number of authors they potentially influenced.

Out-Degree (how many influenced):

```
out_degree = edges_df.groupBy("author1") \
    .count() \
    .withColumnRenamed("count", "out_degree")
```

```
>>> out_degree = edges_df.groupBy("author1") \
...     .count() \
...     .withColumnRenamed("count", "out_degree")
>>> out_degree.show()
+--------------------+----------+
|             author1|out_degree|
+--------------------+----------+
|        Henry Lawson|        13|
|       Lewis Carroll|       255|
|     Robert Nemiroff|        26|
|The National Atom...|        13|
|   Henry H. Snelling|        13|
|   Robert W. Service|       119|
|         Jules Verne|       255|
|       Mark E. Laxer|       113|
|David Herbert Law...|       177|
| James Branch Cabell|        13|
|United States Bur...|       286|
|         Leo Tolstoy|       177|
|  Samuel D. Humphrey|        13|
|Edna St. Vincent ...|        13|
| Adam Lindsay Gordon|        13|
|Franklin Delano R...|        13|
|               Plato|        13|
|Charles Dodgson, ...|       162|
|         H. G. Wells|       432|
|Edgar Rice Burroughs|       720|
+--------------------+----------+
only showing top 20 rows
```

In-Degree (how many influenced them):

```
in_degree = edges_df.groupBy("author2") \
    .count() \
    .withColumnRenamed("count", "in_degree")
```

```
>>> in_degree = edges_df.groupBy("author2") \
...     .count() \
...     .withColumnRenamed("count", "in_degree")
>>> in_degree.show()
+--------------------+---------+
|             author2|in_degree|
+--------------------+---------+
|        Henry Lawson|       66|
|       Lewis Carroll|       45|
|   Franz Josef Haydn|       56|
|     Robert Nemiroff|      132|
|The National Atom...|       66|
|   Henry H. Snelling|       66|
|   Robert W. Service|      178|
|         Jules Verne|      130|
| James Branch Cabell|       66|
|       Mark E. Laxer|       39|
|David Herbert Law...|       12|
|      Giuseppe Salza|       56|
|United States Bur...|       14|
|        Joyce Kilmer|       56|
|         Leo Tolstoy|       12|
|  Samuel D. Humphrey|       66|
|    Geoffrey Chaucer|       56|
|Edna St. Vincent ...|       66|
| Adam Lindsay Gordon|       66|
|Franklin Delano R...|       66|
+--------------------+---------+
only showing top 20 rows
```

2. Identify the top 5 authors with the highest in-degree and the top 5 authors with the highest out-degree.

Top 5 by Out-Degree:

```
>>> out_degree.orderBy(col("out_degree").desc()).show(5)
+--------------------+----------+
|             author1|out_degree|
+--------------------+----------+
|        Thomas Hardy|       828|
|Robert Louis Stev...|       817|
|Edgar Rice Burroughs|       720|
|         Henry James|       613|
|Frances Hodgson B...|       551|
+--------------------+----------+
only showing top 5 rows
```

Top 5 by In-Degree:

```
>>> in_degree.orderBy(col("in_degree").desc()).show(5)
+--------------------+---------+
|             author2|in_degree|
+--------------------+---------+
|Robert Louis Stev...|     1911|
|Edgar Rice Burroughs|      941|
|  Arthur Conan Doyle|      615|
|       L. Frank Baum|      552|
|Richard Harding D...|      512|
+--------------------+---------+
only showing top 5 rows
```

PART 4:

Discuss how you chose to represent the influence network in Spark (e.g., RDD of tuples, DataFrame). What are the advantages and disadvantages of your chosen representation?

I represented the author influence network as a **Spark DataFrame** containing directed edges of the form: (author1, author2).

Where:

- author1 → potentially influenced
- author2 → potentially influenced by
- Each row represents one directed influence relationship.

Why I Chose DataFrame (Instead of RDD)?

Advantages:

1. Optimized Execution- Spark SQL optimizer (Catalyst) improves join and aggregation performance.
2. Easy Degree Computation - In-degree and out-degree can be computed easily using: groupBy().count()
3. Scalable –
    - Works efficiently for large datasets.
    - Supports distributed joins.
4. Readable & Structured –
    - Schema-based.
    - Easier debugging and filtering.

Disadvantages:

1. Not a Native Graph Structure - No built-in graph traversal (e.g., PageRank, shortest path) unless using GraphFrames.
2. Self-Join Can Be Expensive - Influence detection requires comparing authors, which may become costly for large datasets.
3. Limited Graph Semantics - Unlike GraphX or GraphFrames, relationships are not automatically treated as graph objects.

Why Not RDD?

RDD of tuples like:

("Jane Austen", "Charles Dickens")

would work, but:

- No query optimization
- Harder aggregation
- More manual coding
- Slower for large joins

I represented the influence network as a Spark DataFrame of directed edges (author1, author2). This approach enables efficient distributed joins and easy degree calculations using group-by operations. While it is scalable and optimized, it lacks native graph-processing capabilities compared to GraphFrames or GraphX.

PART 5:

How does the choice of the time window (X) affect the structure of the influence network?
What are the limitations of this simplified definition of influence?

1. How does the choice of time window (X years) affect the network?

Assume we define influence as:
Author A influences Author B if B published within **X years** after A.

If X is Small (e.g., 2–5 years)

- Fewer influence edges

- Network becomes **sparser**
- Only very close temporal relationships are captured
- Likely lower in-degree and out-degree
- May miss long-term literary influence

Example:
If Jane Austen (1813) and Dickens (1837) → 24 years apart
With X = 5 → no edge
With X = 30 → edge appears

If X is Large (e.g., 30–50 years)

- Many more edges
- Network becomes **denser**
- Authors appear highly connected
- May create unrealistic influence links
- Can inflate in-degree/out-degree

Too large X → almost everyone influences everyone.

So the time window directly controls:

- Edge density
- Connectivity
- Degree distribution
- Centrality of authors

2. Limitations of This Simplified Influence Definition

This definition is **very naive**. It assumes:

Time proximity = influence

But in reality, influence is much more complex.

Major Limitations:

1. No Content Similarity - Two authors close in time may write completely different genres.
2. No Citation or Historical Evidence –

   It doesn't check whether:

   - Author B read Author A
   - Author B referenced Author A
   - There is stylistic similarity

3. Ignores Long-Term Influence - Some authors influence writers decades later (e.g., Shakespeare).
4. False Positives - Just because books were published close in time doesn't mean influence exists.

5. Ignores Multiple Works - An author may publish many books over time — using only one release date oversimplifies.

Better Real-World Approach:

A stronger influence model could combine:

- Time proximity
- Text similarity (TF-IDF cosine similarity)
- Citation/reference analysis
- Genre similarity
- Historical records

In short:

A smaller time window produces a sparse network with fewer influence edges, while a larger window creates a denser network with many connections. However, defining influence purely based on temporal proximity is limited because it ignores textual similarity, citations, genre, and historical evidence. Time alone does not guarantee actual influence.