设计思路：

对于本实验，其本质是对csv文件进行读取，并且按照实验取出相应的列进行计数并且排序。但需要注意headline列中可能带有逗号，因此直接使用逗号分割再进行相应列的读取可能导致读取到非目标列，因此再两个任务中对recordreader返回的value值分别处理：任务1中选取最后一列（股票列为最后一列），任务二循环遍历split后的字符串循环范围为1~len(string)-2,即可获取所有标题列中的内容。

按照以下思路分别进行设计

Stockcount：

```
public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String line = value.toString();

            // 使用逗号分隔行，但股票代码是最后一列，前面的字段可能包含逗号
            String[] fields = line.split(",", -1);

            if (fields.length > 0) {
                String stockCodeValue = fields[fields.length - 1].trim(); // 获取
最后一列作为股票代码
                if (!stockCodeValue.isEmpty()) {
                    stockCode.set(stockCodeValue);
                    context.write(stockCode, one); // 输出<股票代码，1>
                }
            }
        }
```

map任务按照要求取出最后一列，利用context.write进行传输即可

```
public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            stockCountMap.put(key.toString(), sum);  // 保存每个股票代码和对应的计数
        }
```

reduce阶段利用put方法将各支股票计数总和进行保存

```
@Override
        protected void cleanup(Context context) throws IOException,
InterruptedException {
            // 使用TreeMap根据值的大小进行降序排序
            TreeMap<String, Integer> sortedMap = new TreeMap<>(new
Comparator<String>() {
                public int compare(String key1, String key2) {
                    int compare =
stockCountMap.get(key2).compareTo(stockCountMap.get(key1)); // 按值降序
                    if (compare == 0) {
                        return key1.compareTo(key2); // 如果值相同，按股票代码字母顺序
升序
                    }
                    return compare;
```

```
                }
            });
            sortedMap.putAll(stockCountMap);  // 将所有元素放入排序的TreeMap

            // 输出排序后的结果
            for (Map.Entry<String, Integer> entry : sortedMap.entrySet()) {
                context.write(new Text(entry.getKey()), new
IntWritable(entry.getValue()));
            }
        }
```

cleanup阶段利用Treemap进行排序，自定义排序方法，利用putAll方法将计数统计到treemap中，进行排序并输出即可

结果为

WEB最后一列为StockCount运行结果

Wordcount:

```
    @Override
    protected void setup(Context context) throws IOException,
InterruptedException {
        // Load stop-word list from command-line argument
        String stopWordsFilePath =
context.getConfiguration().get("stopwords.file");
        FileSystem fs = FileSystem.get(context.getConfiguration());
        FSDataInputStream inputStream = null;
        BufferedReader br = null;

        try {
            inputStream = fs.open(new Path(stopWordsFilePath));
            br = new BufferedReader(new InputStreamReader(inputStream));
            String line;
            while ((line = br.readLine()) != null) {
                stopWords.add(line.trim().toLowerCase());
            }
        } finally {
            IOUtils.closeStream(br);
            IOUtils.closeStream(inputStream);
        }
    }
```

先覆写setup类，利用InputStreamReader读取HDFS中的停词文件，将停词存入Set中以便后续去除

```
    @Override
    protected void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
        // Split CSV file
        String[] fields = value.toString().split(",");

        // Check if there are enough fields
```

```
        if (fields.length > 2) {  // 至少需要有 3 列
            // 拼接剩余列内容，去掉第一列和最后一列
            StringBuilder headlineBuilder = new StringBuilder();
            for (int i = 1; i < fields.length - 2; i++) {
                if (i > 1) {
                    headlineBuilder.append(",");  // 在字段之间添加逗号
                }
                headlineBuilder.append(fields[i].trim());
            }

            // 转换为小写并移除标点符号
            String headline =
headlineBuilder.toString().toLowerCase().replaceAll("[^a-zA-Z\\s]", "");

            // 使用 StringTokenizer 进行分词
            StringTokenizer tokenizer = new StringTokenizer(headline);
            while (tokenizer.hasMoreTokens()) {
                String token = tokenizer.nextToken();
                if (!stopWords.contains(token)) {
                    word.set(token);
                    context.write(word, one); // Emit <word, 1>
                }
            }
        }
    }
```

map类中，按前面的设计思路取出headline部分，进行分词，过滤set中的停词文件进行写出即可

```
@Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            wordCountMap.put(key.toString(), sum);
        }
```

reduce类只需对相同key的单词进行总和统计即可

```
protected void cleanup(Context context) throws IOException, InterruptedException
{
    // Priority queue to maintain top 100 frequent words
    PriorityQueue<Map.Entry<String, Integer>> topWords = new PriorityQueue<>(
        new Comparator<Map.Entry<String, Integer>>() {
            @Override
            public int compare(Map.Entry<String, Integer> o1, Map.Entry<String,
Integer> o2) {
                return o1.getValue().compareTo(o2.getValue());  // Ascending
order by count
            }
        });
```

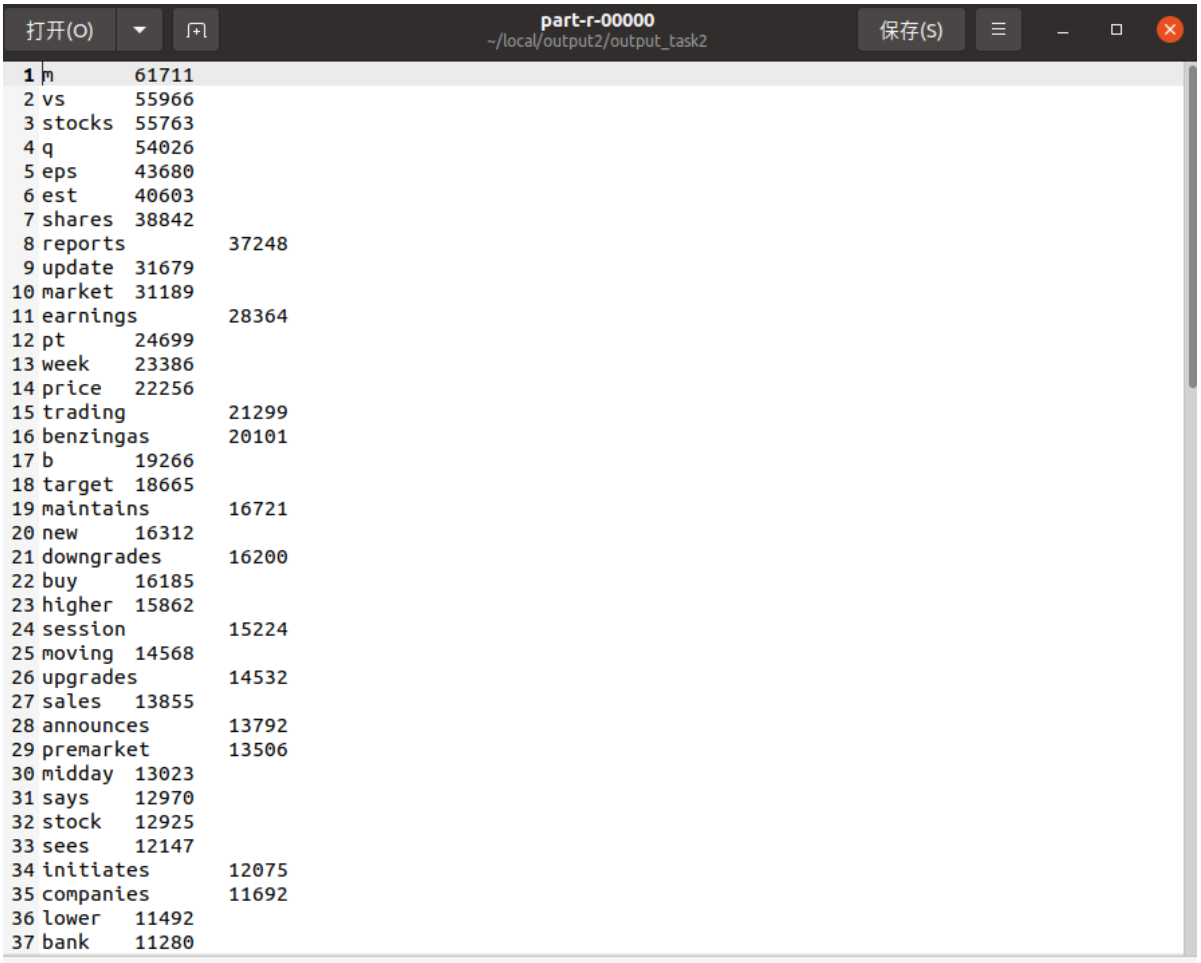
```
    // Add to priority queue and maintain size <= 100
    for (Map.Entry<String, Integer> entry : wordCountMap.entrySet()) {
        topWords.offer(entry);
        if (topWords.size() > 100) {
            topWords.poll();  // Remove smallest entry if size exceeds 100
        }
    }

    // Collect top words to a list to sort them later
    List<Map.Entry<String, Integer>> sortedTopWords = new ArrayList<>(topWords);
    // Sort in descending order by frequency
    sortedTopWords.sort((o1, o2) -> o2.getValue().compareTo(o1.getValue()));

    // Output the top 100 words in descending order of frequency
    for (Map.Entry<String, Integer> entry : sortedTopWords) {
        context.write(new Text(entry.getKey()), new
IntWritable(entry.getValue()));
    }
}
    }
```

cleanup类用于排序以及输出，利用优先队列维护词频为前100的单词，并最终利用context.write输出。

输出结果为：




```
m       61711
vs      55966
stocks  55763
q       54026
eps     43680
est     40603
shares  38842
reports         37248
update  31679
market  31189
earnings        28364
pt      24699
week    23386
price   22256
trading         21299
benzingas       20101
b       19266
target  18665
maintains       16721
new     16312
downgrades      16200
buy     16185
higher  15862
session         15224
moving  14568
upgrades        14532
sales   13855
announces       13792
premarket       13506
midday  13023
says    12970
stock   12925
sees    12147
initiates       12075
companies       11692
lower   11492
bank    11280
```

作业 | Namenode information | All Applications

localhost:8088/cluster

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Used Resources	Total Resour
18	0	0	18	0	<memory:0 B, vCores:0>	<memory:8 GB, vCores:8>

Cluster Nodes Metrics

Active Nodes	Decommissioning Nodes	Decommissioned Nodes	Lost Nodes	
1	0	0	0	0

Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Capacity Scheduler	[memory-mb (unit=Mi), vcores]	<memory:1024, vCores:1>	<memory:8192, vCores:4>

Show 20 entries

ID	User	Name	Application Type	Application Tags	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU VCores
application_1729174327876_0018	user	Word Count with Top 100 Frequent Words	MAPREDUCE		default	0	Thu Oct 17 23:44:41 +0800 2024	Thu Oct 17 23:44:42 +0800 2024	Thu Oct 17 23:44:57 +0800 2024	FINISHED	SUCCEEDED	N/A	N/A