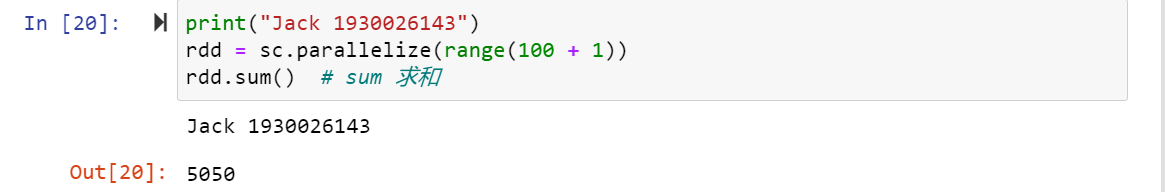
**Lab7 – Spark RDD Operations**

薛劭杰 1930026143

LabS02\_1\_RDD\_Operations

1. Transformation 

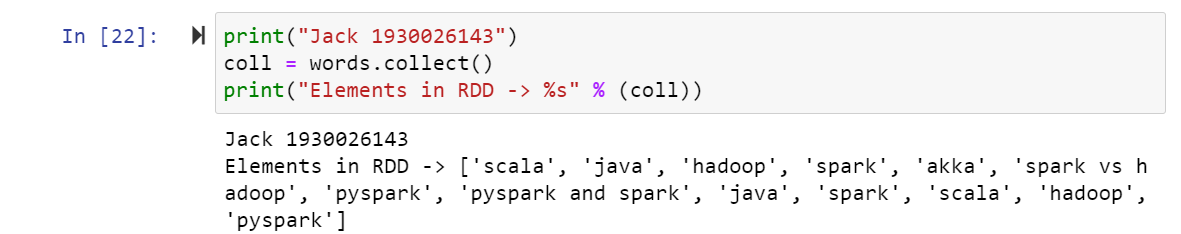
Operation: sum of 1+2+3+4...+100



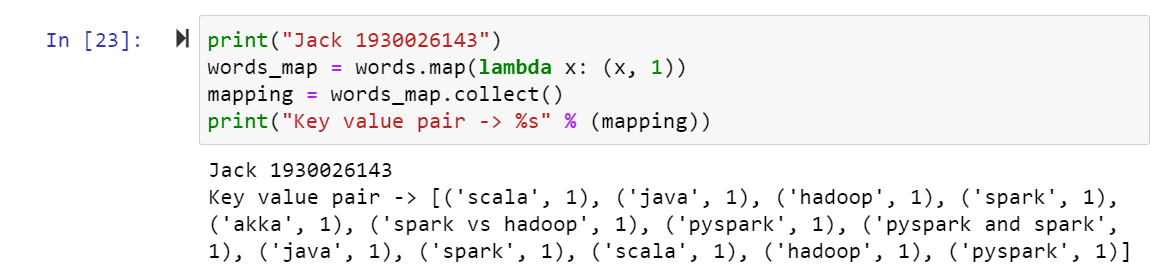
1. parallelize array of words and count



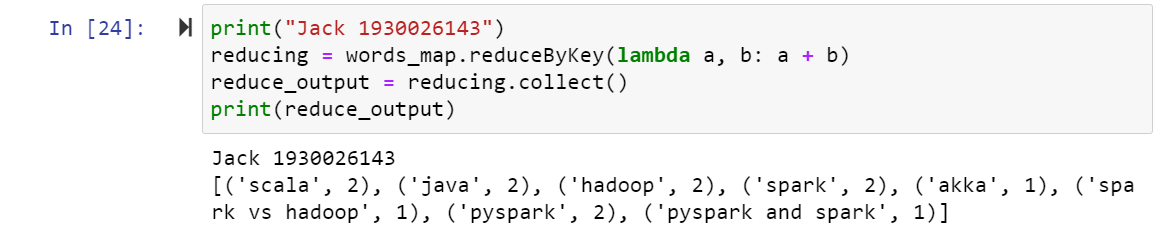
1. use collect operation to get elements from the RDD¶



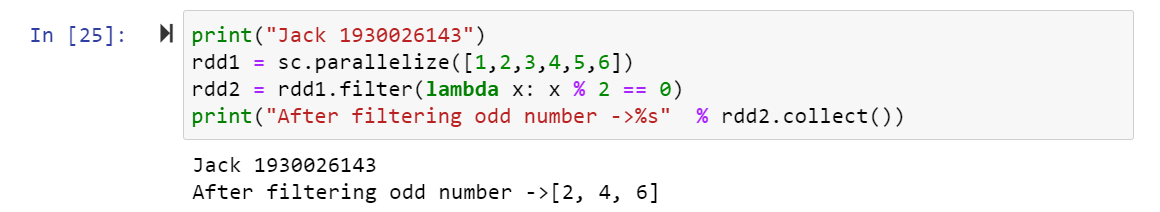
1. map: is used to apply the transformation function (lambda) on every element of RDD/DataFrame and returns a new RDD



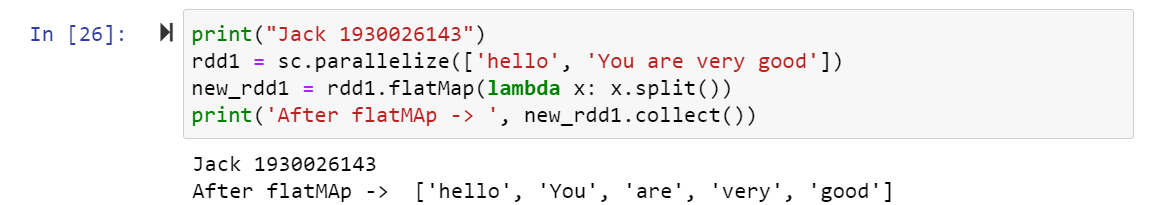
1. reduceByKey(): Merge the values for each key using an associative and commutative reduce function. In this case, the key word will collect together and the value is



1. Filter: Select the even number in the RDD



1. flatMap: Decompose a group of sentences into a single string

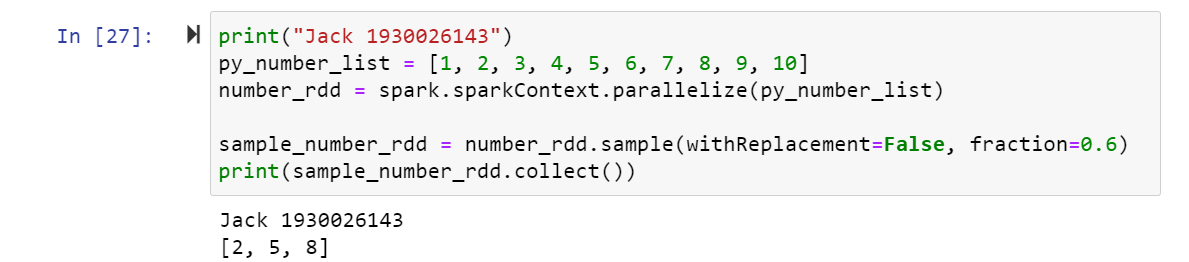


1. Sample: Returns a sampled subset of this DataFrame.

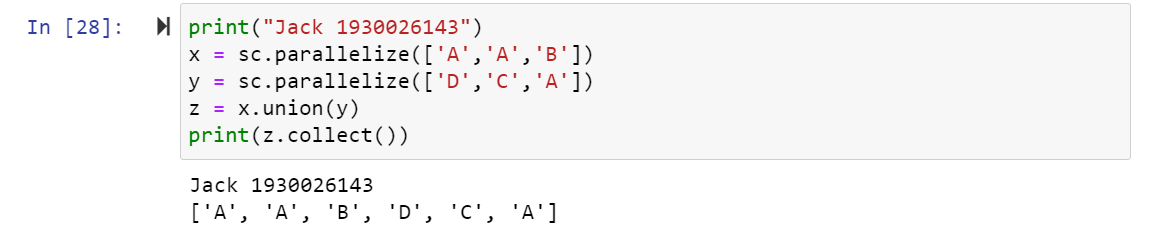
: Sample with replacement or not (default False).

: Fraction of rows to generate, range.

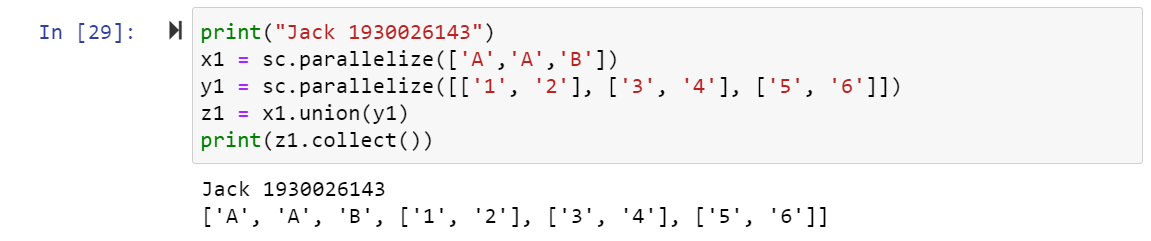
: Seed for sampling (default a random seed)



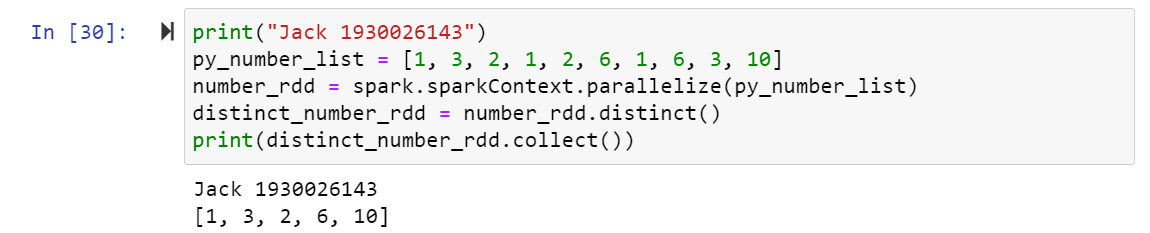
1. Union: It is not union set but just a connection of two RDD.



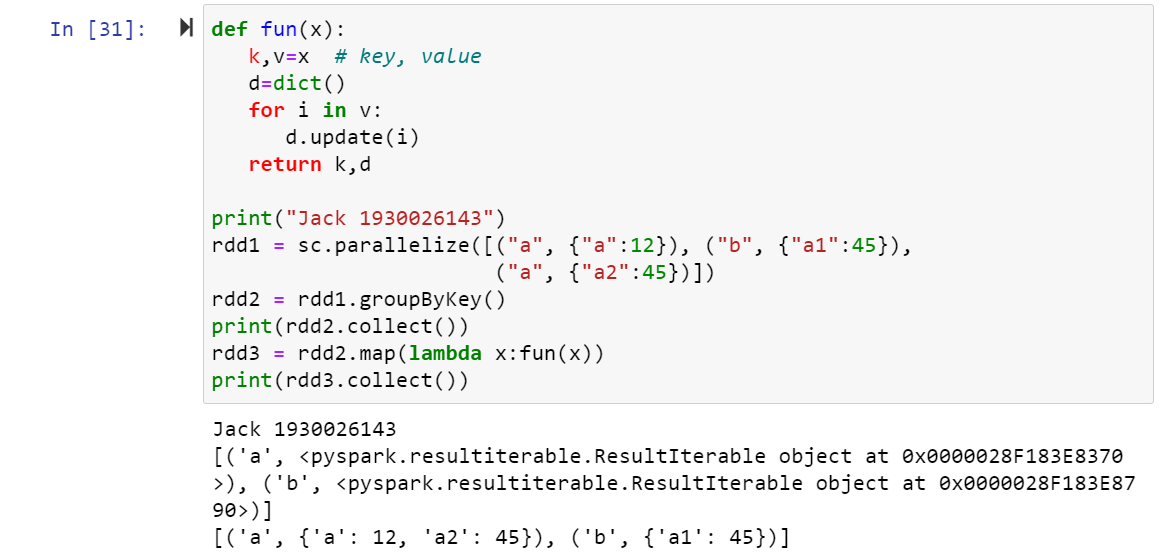
Different type union:



1. Distinct: get rid of the duplicated elements.



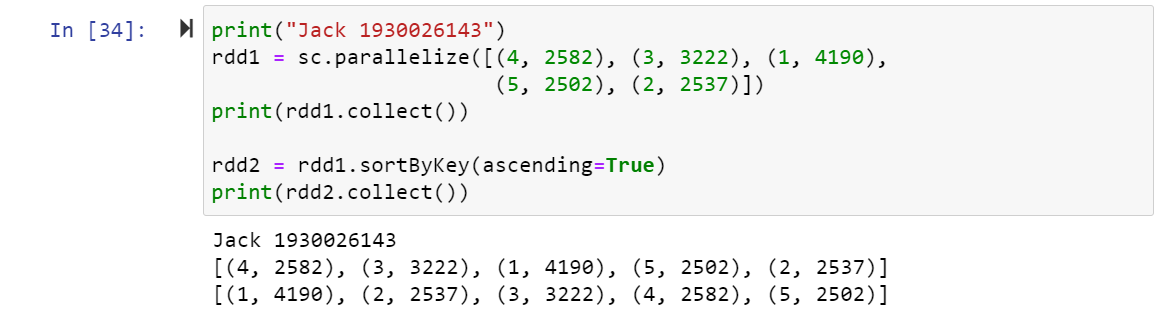
1. groupByKey: Firstly, groupbykey and use map() to transform them in the fun() format



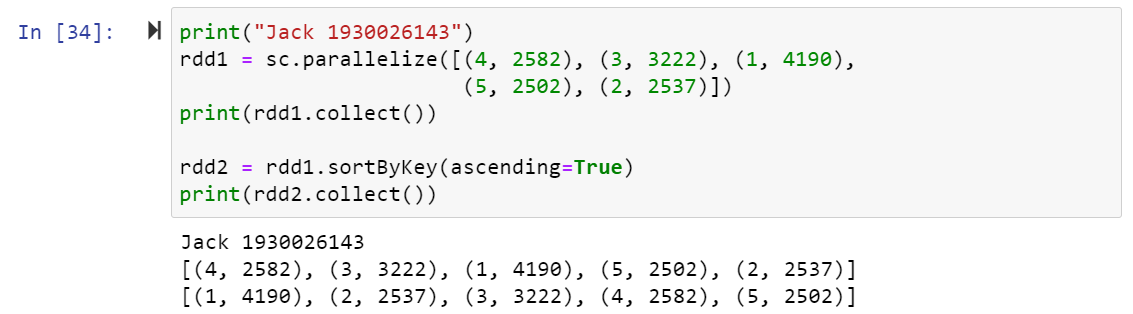
1. reduceByKey: the first element is the key and the second one is the value. The function will group by the key and transform the value in new format you want to set.



There alse can be a function as parameter.



1. sortByKey: the first element is the key and sort it in order.

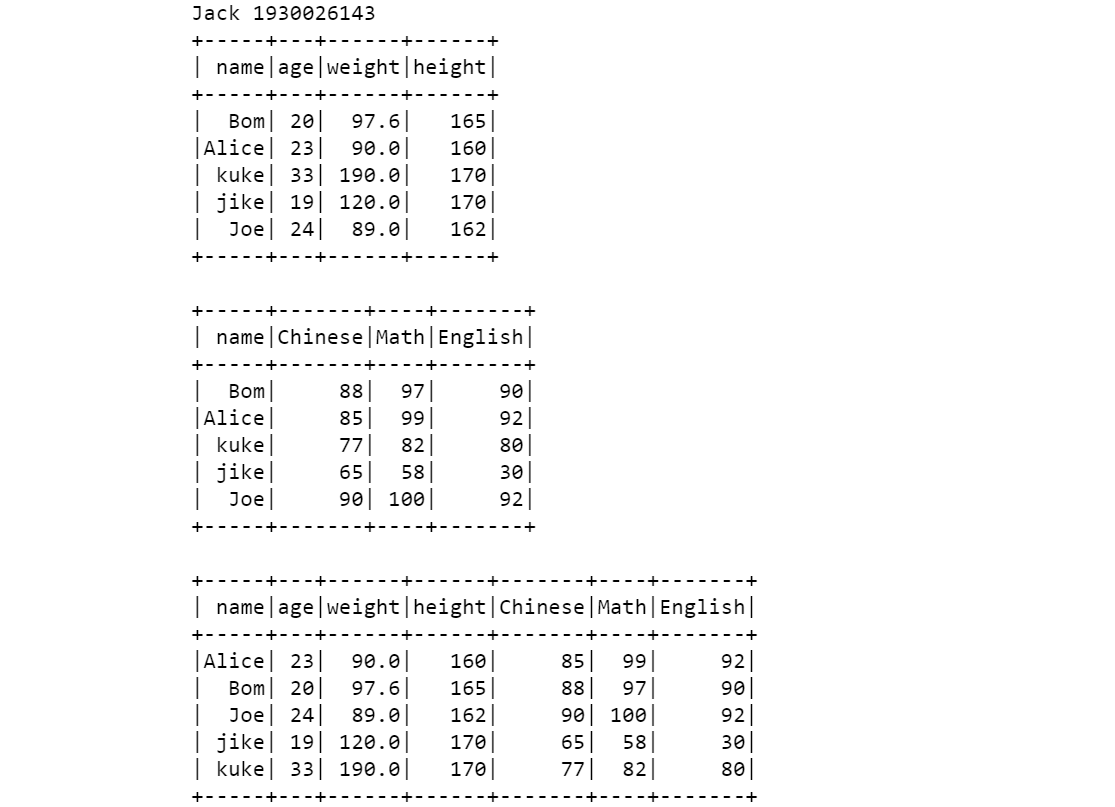


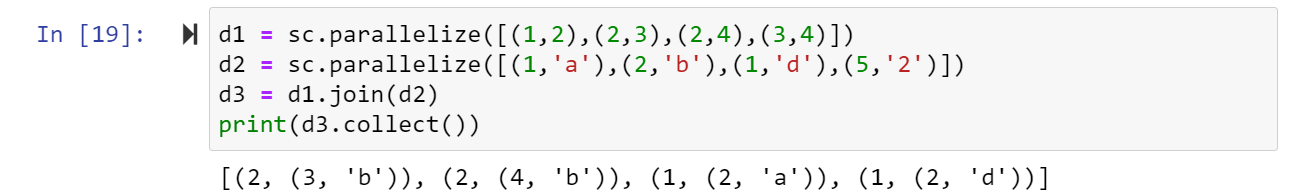
1. Joining elements containing matching keys. This fuction just like a special case of the map() function. It add the key in a sequence type which likes tuple.

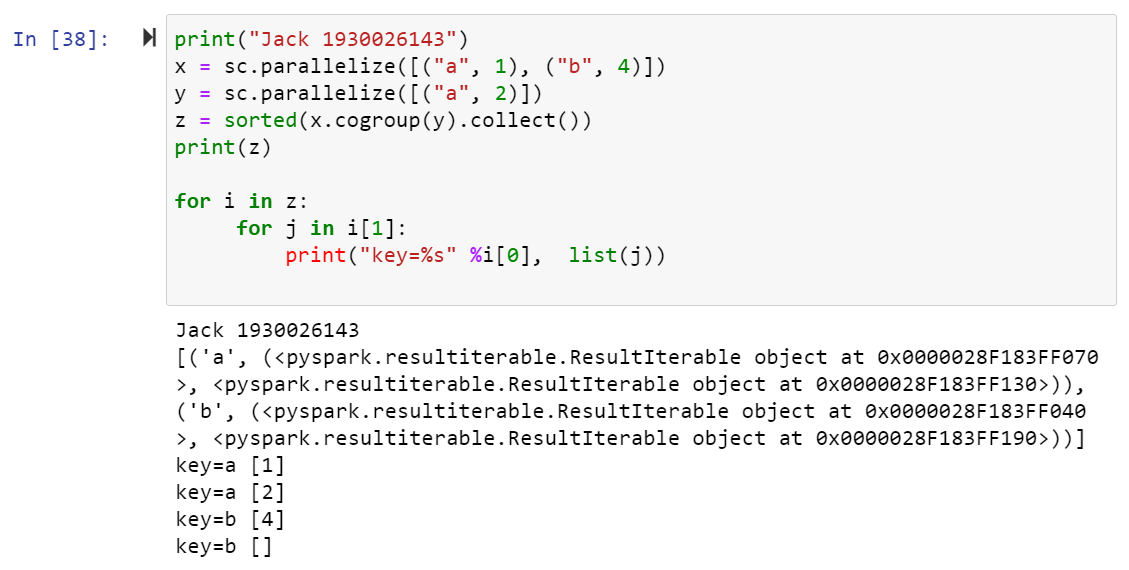


function can apply to two spark.dataFrame with a common attribute, merge it to a dataFrame.

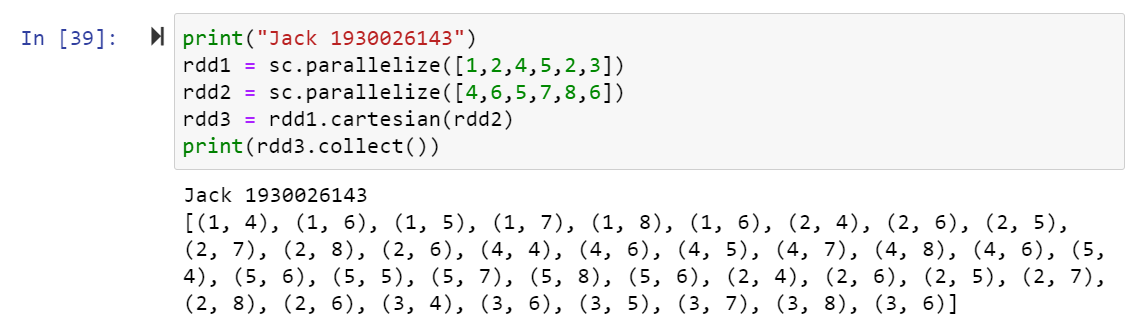




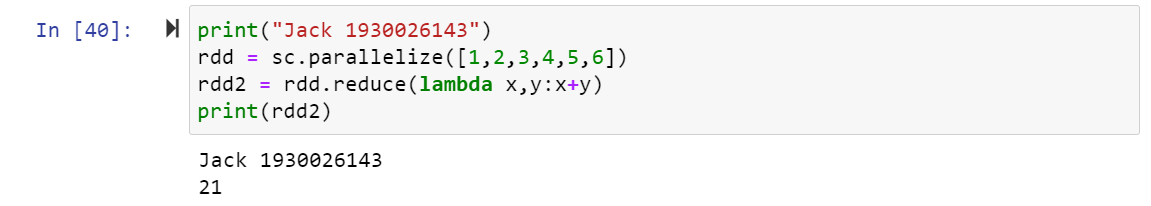
You can find the elementsin two RDD must be one to one correspondence.



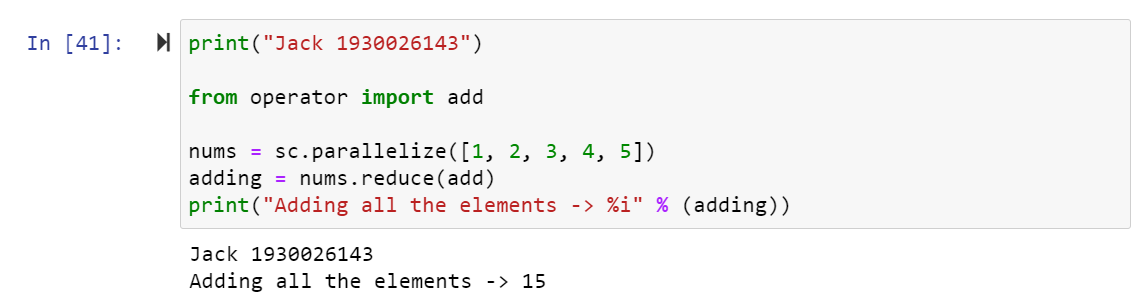
1. Cartesian: Cartesian product



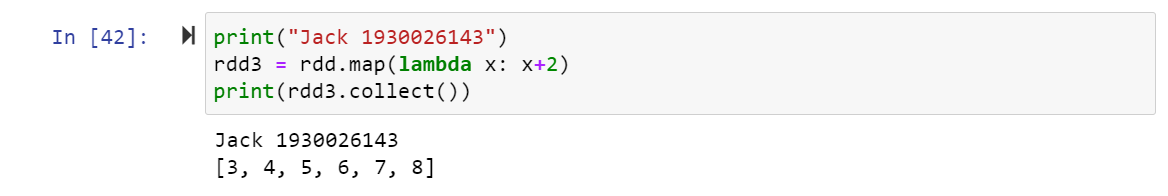
1. Action reduce(): : get a answer (value) after action.



The result just same as the above one.

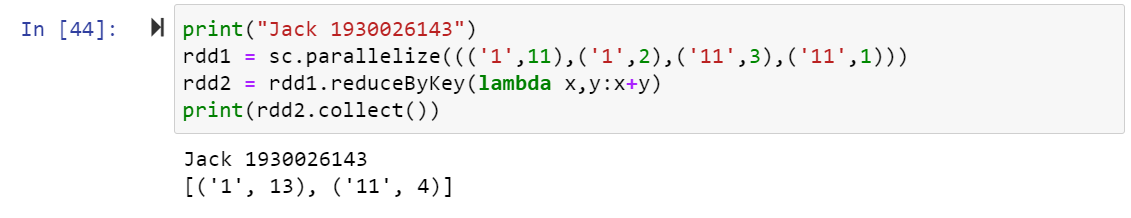


1. Action collect(): get a new rdd after action.

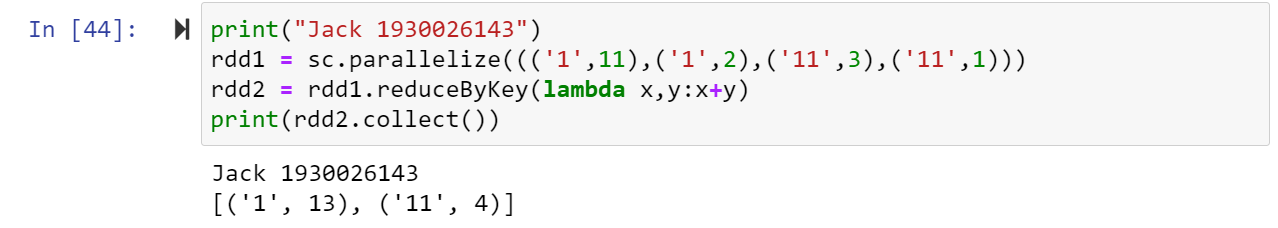


1. : the length of the rdd.

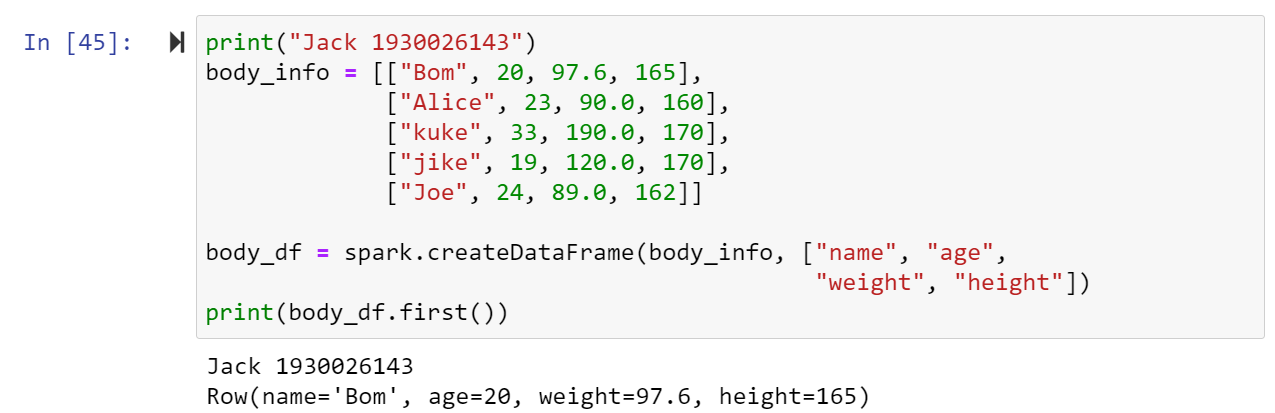
: the number of each value in a directory.



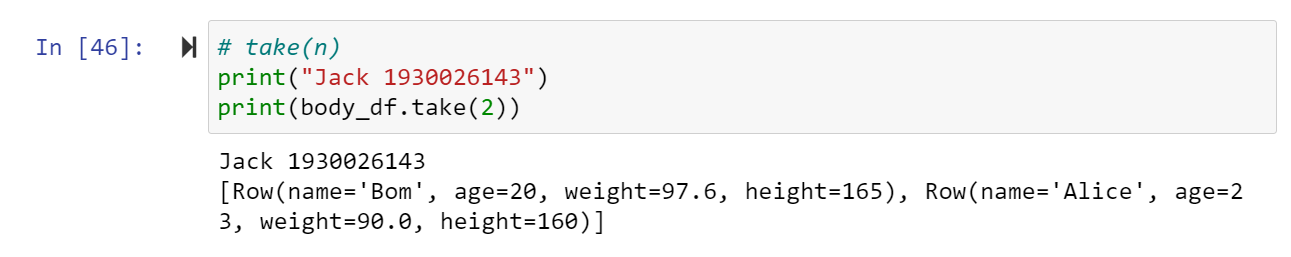
1. Action reduceByKey:



1. Action : take the first row information.

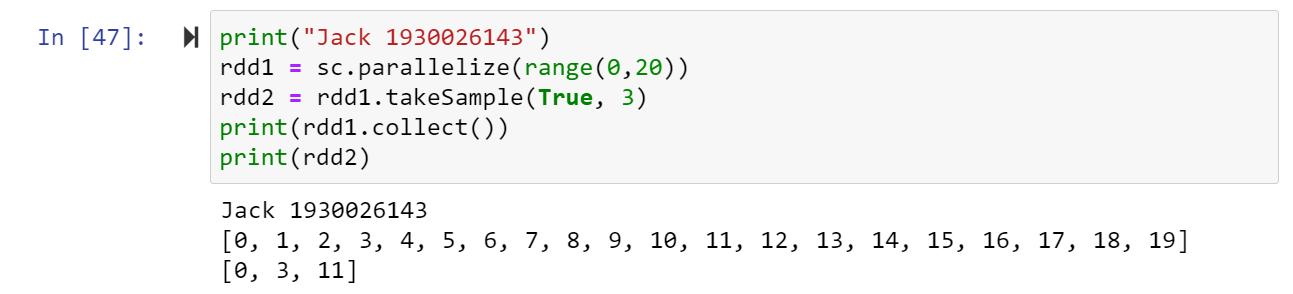


: select the first n row in thr spark.dataFrame.

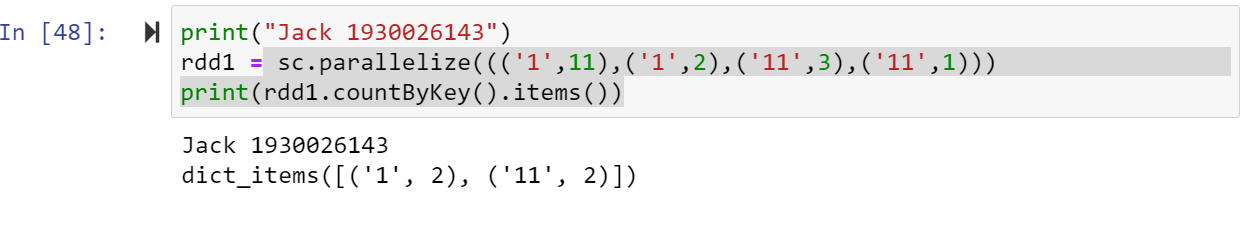


1. (withReplacement, fraction, seed):

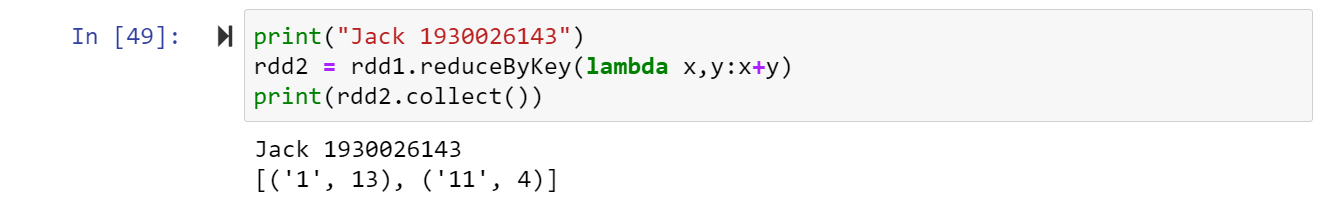
Random select some sample in the rdd.



1. : count the number of values for a same key.

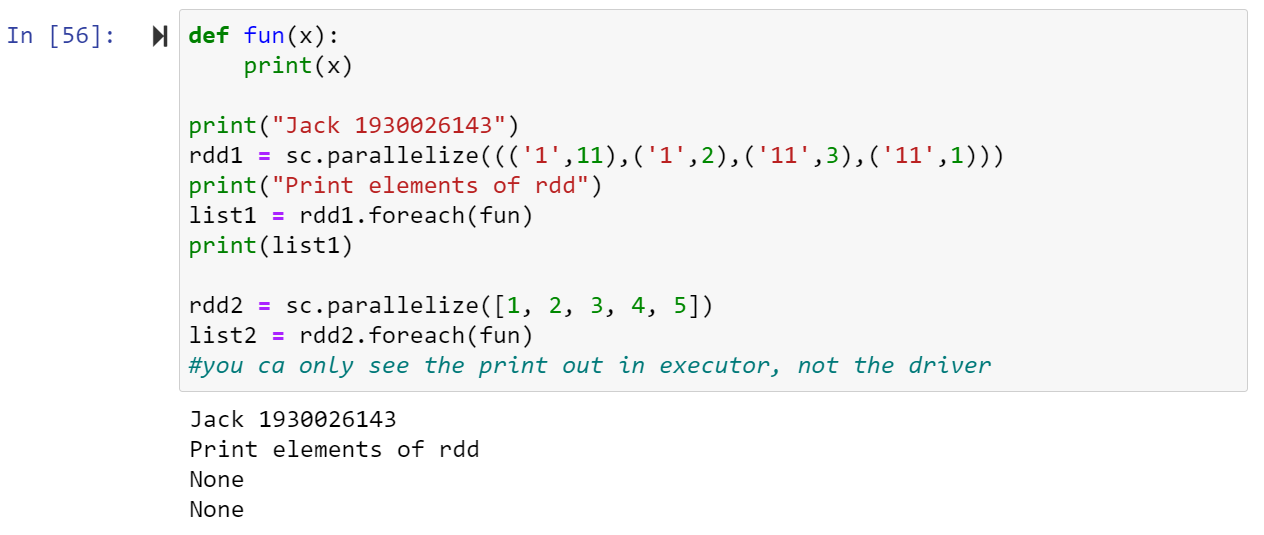


1. reduceByKey(): The format of its parameter is different from above one. It can be a list contains some tuples with key and value.

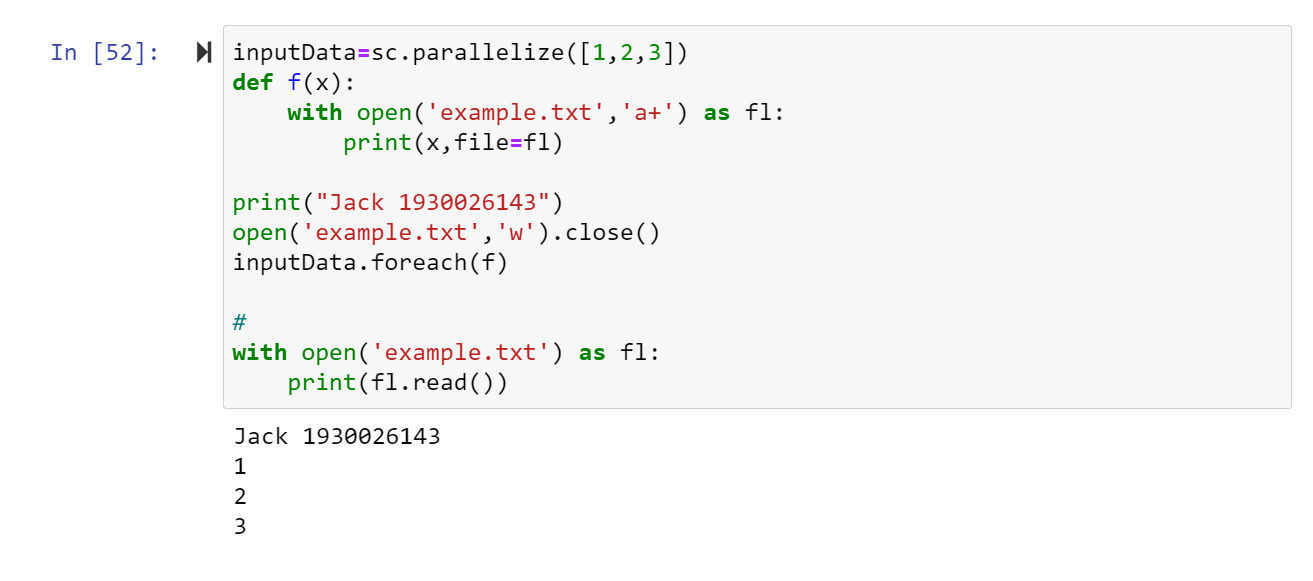




1. foreach(): you can only see the print out in executor, not the driver



Save in a file and read will see the result.



You can deal with the data, then output(save) the file.



1. saveAsTextFile and saveAsSequenceFile



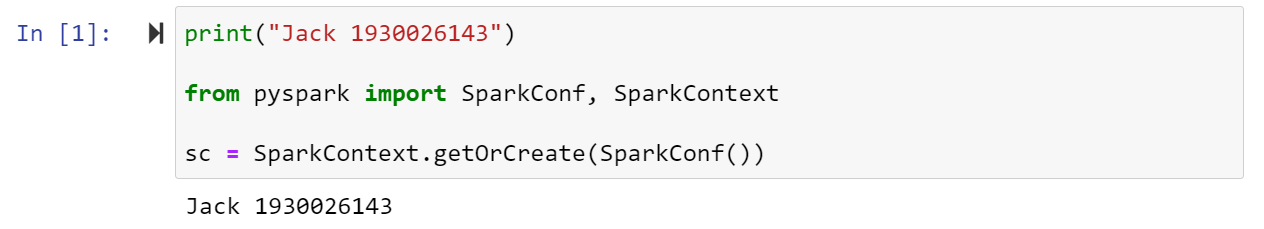
Check the saved file in the local path and if you can see this result, that means it save successfully.





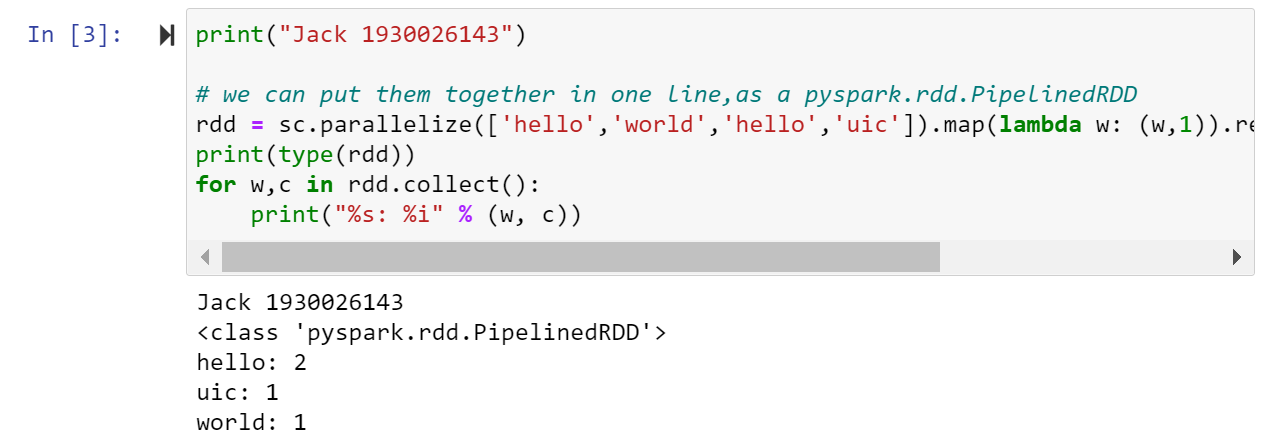
LabS02\_2\_WordCount

1. Implementing a Multi-File Word Count

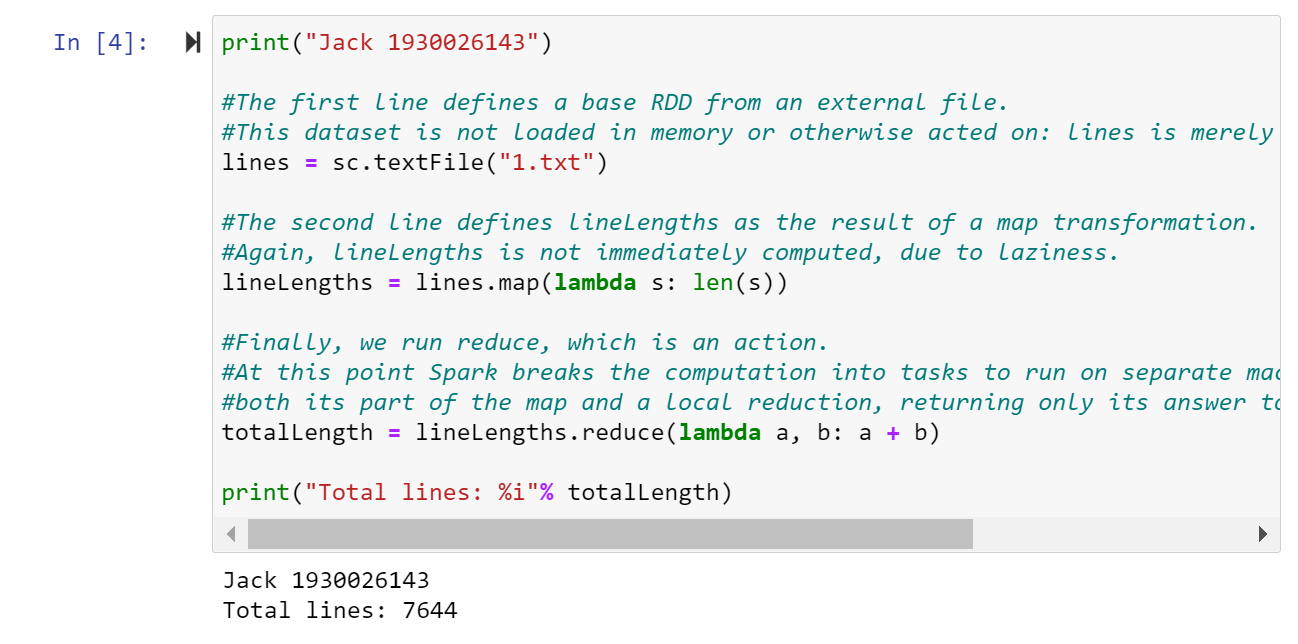


1. Do word count operation in the spark: Spark uses MapReduce on RDD to do word count.

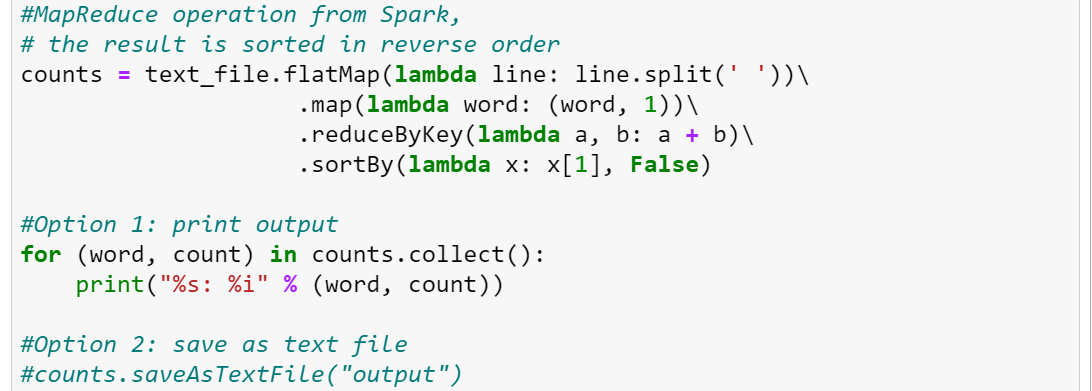


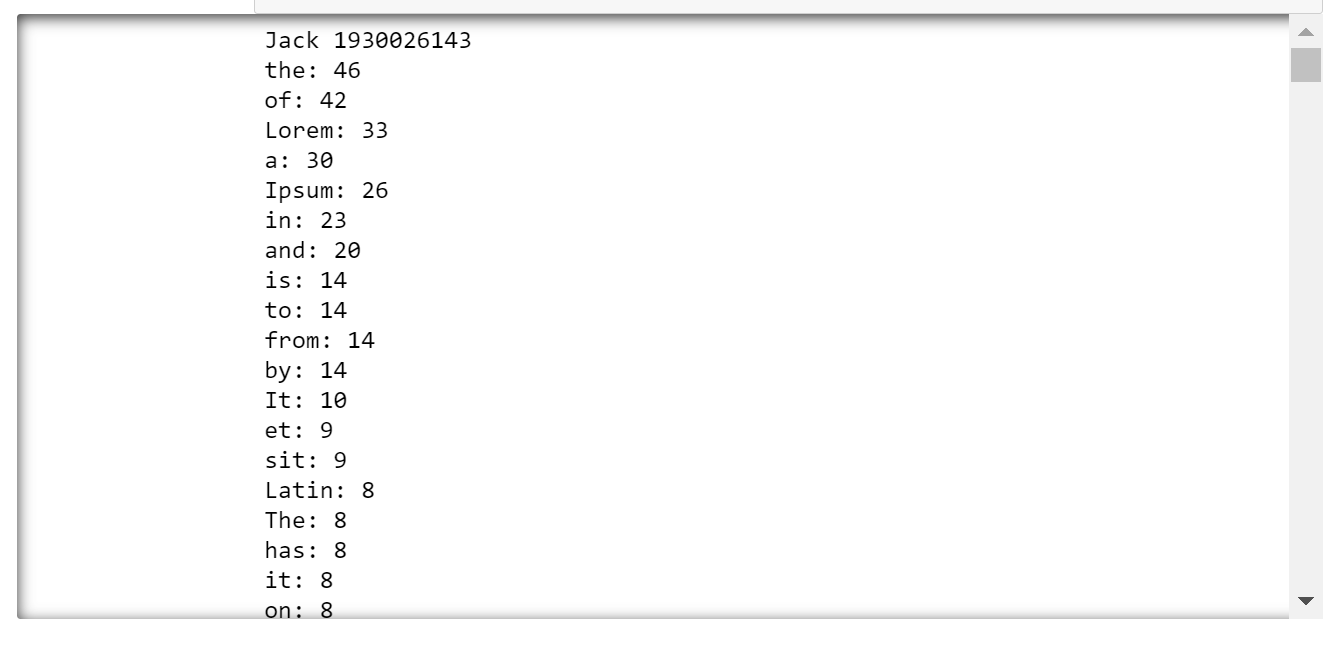


1. Text file can also be processed as RDD in Spark

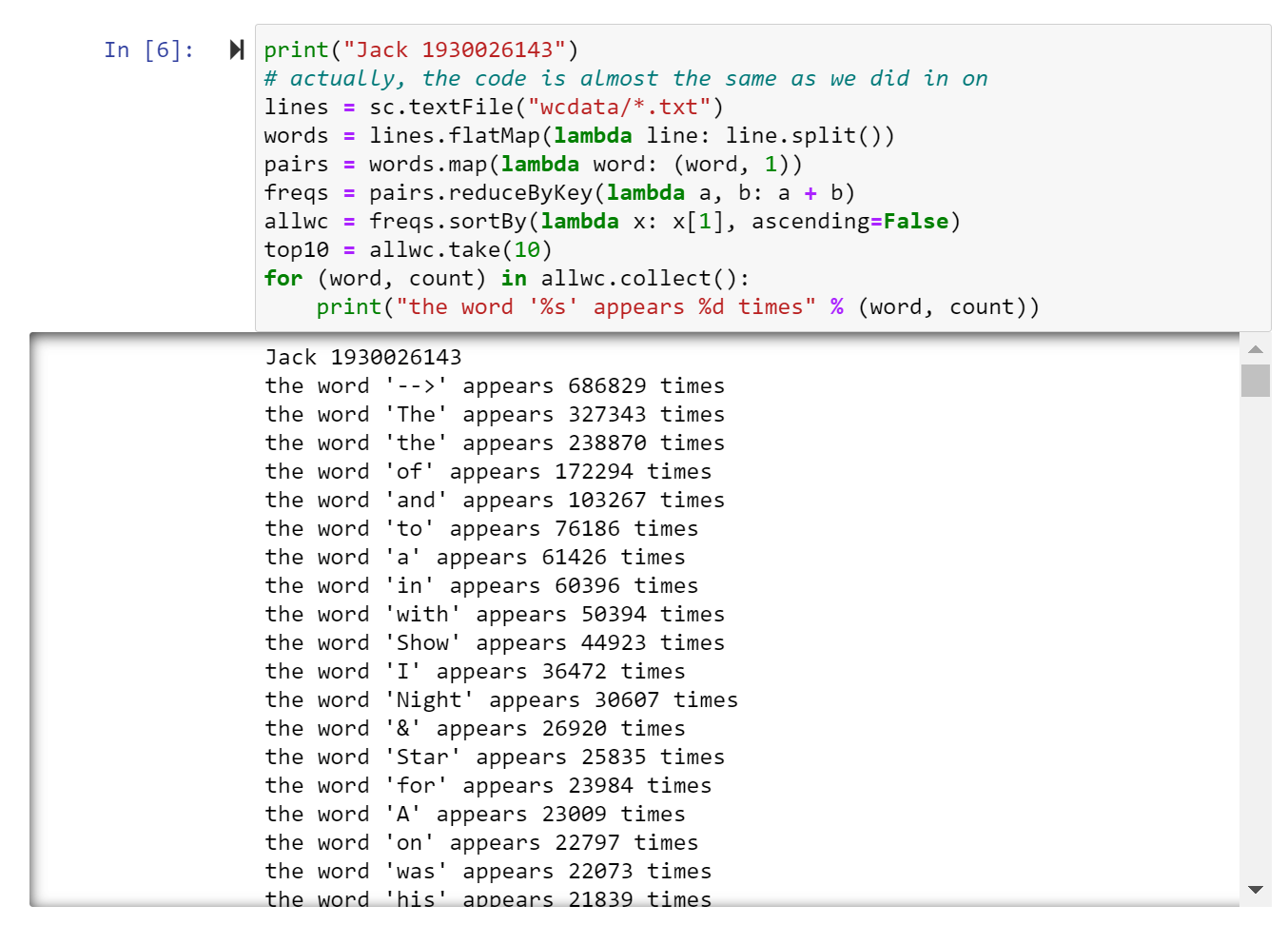








1. Then, we implement the word-counting program for multiple files.

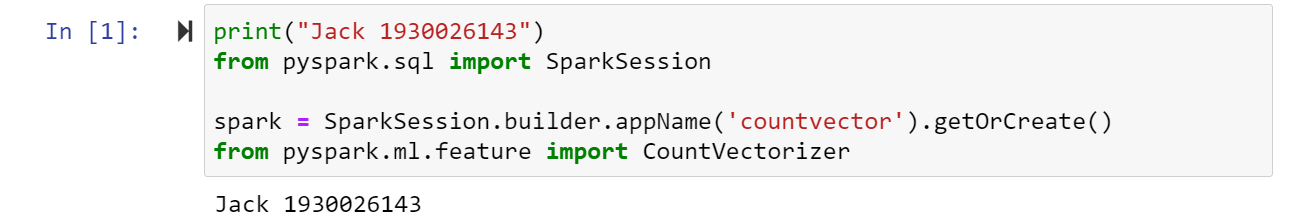


LabS02\_3\_CountVectorization

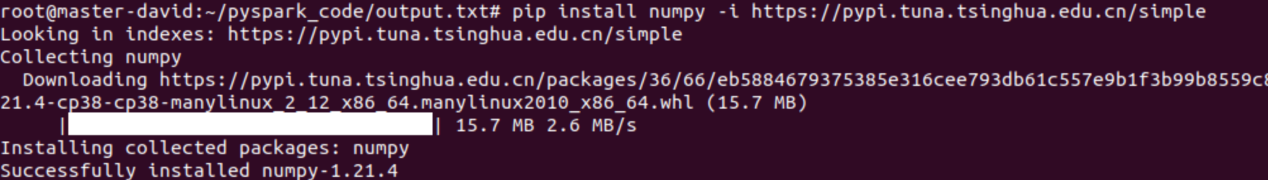
can be used as an Estimator to extract the vocabulary, and generates a CountVectorizerModel.

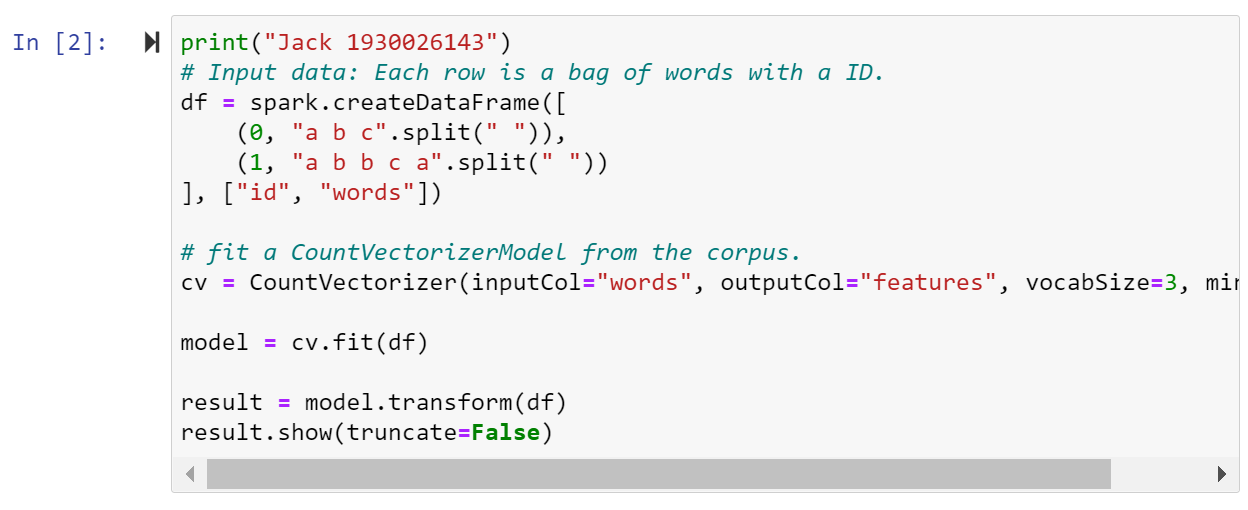
The model generates a sparse representation of the document through a vocabulary, which can then be passed to other algorithms.

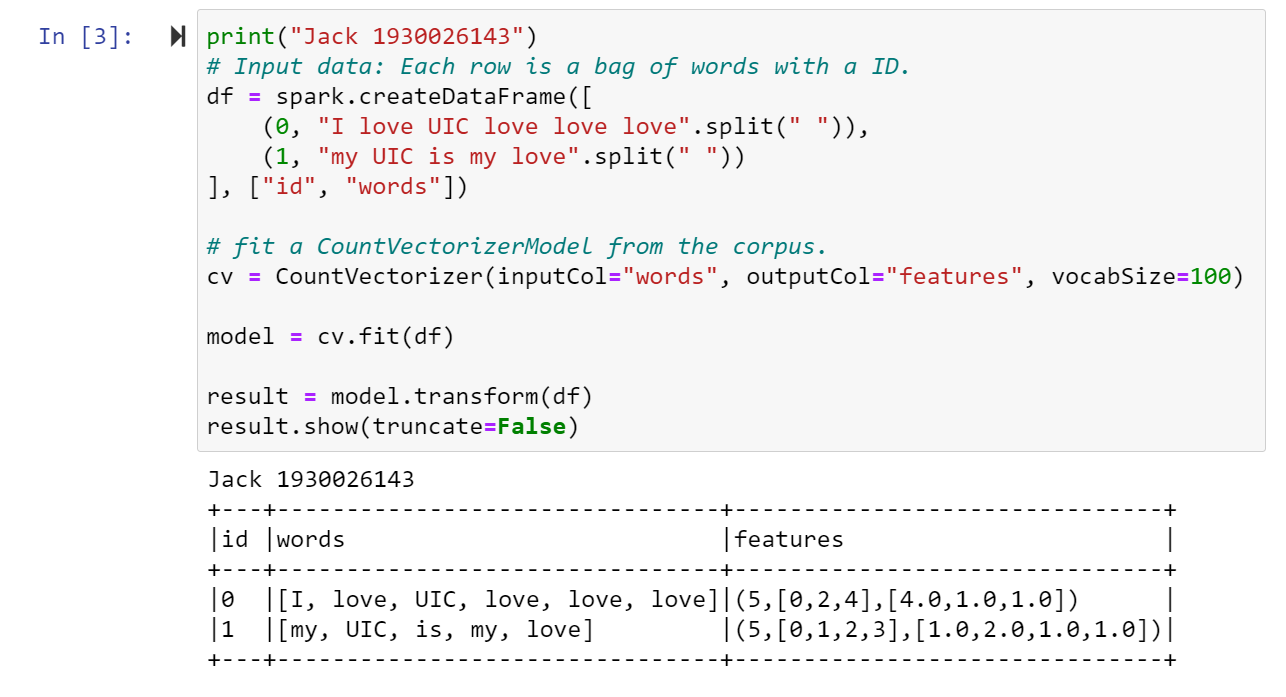
When you execute this cod to lead the package, you may get some error because the version of you numpy is too low.



So, you must update it first.



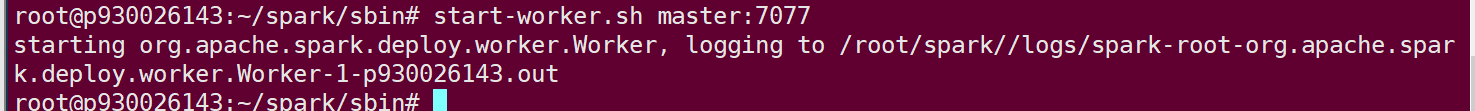


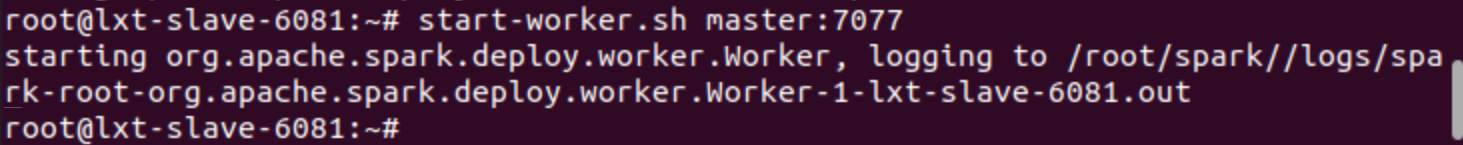


Group Cluster:

The first project:

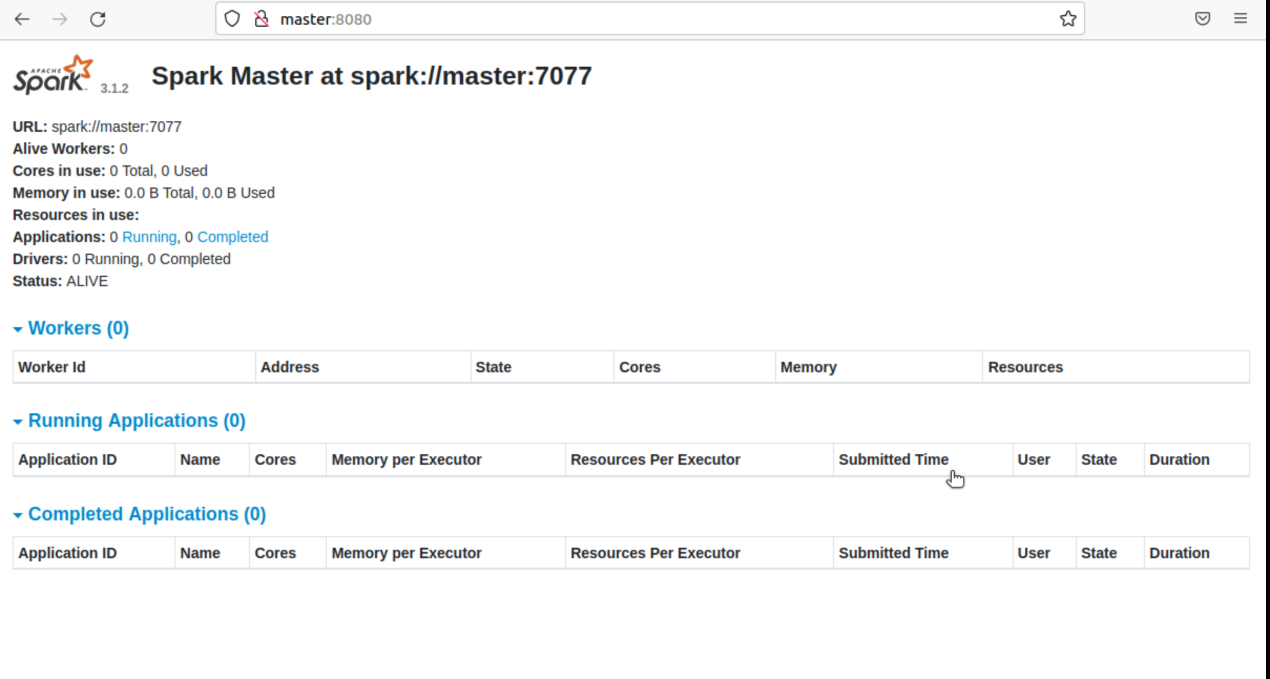
Start the spark cluster you finish in the lab6.



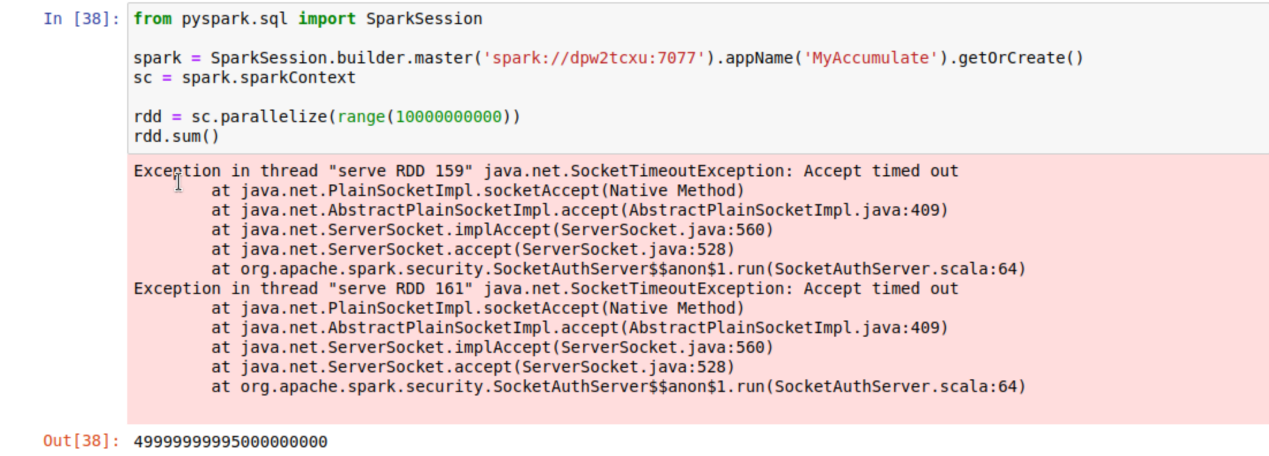


At the master machine, you can just write down the command:

start-master.sh



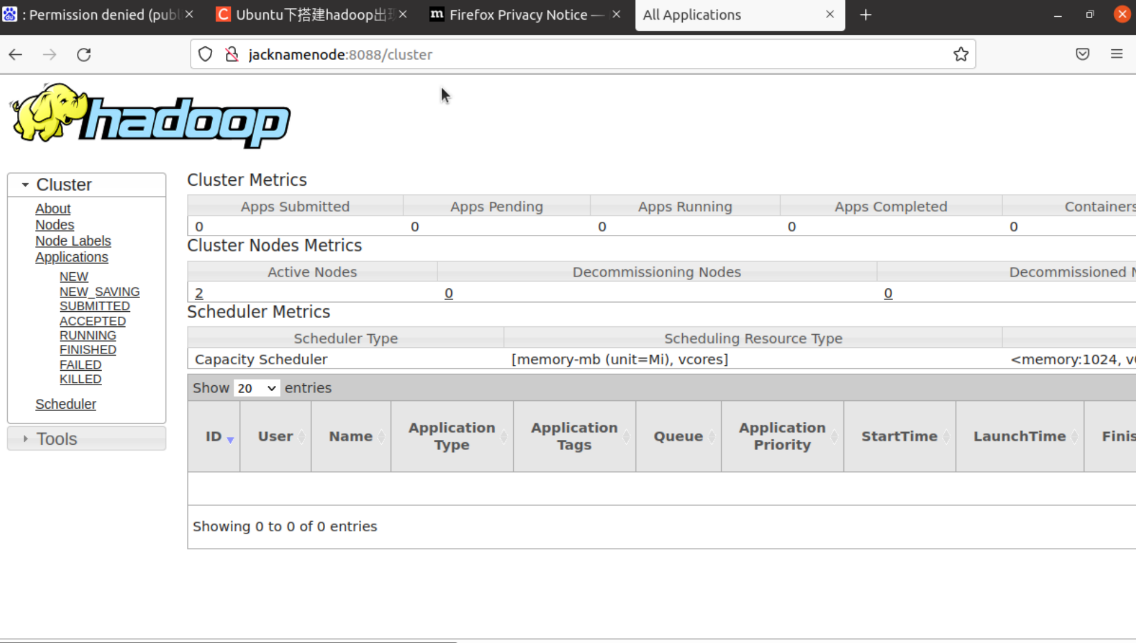
Then after starting the spark cluster, run the code. The exception is right because the digit is too large, it will took a quite long time.



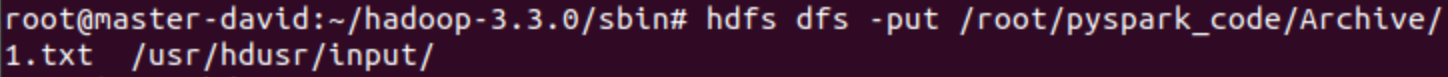
The second project:

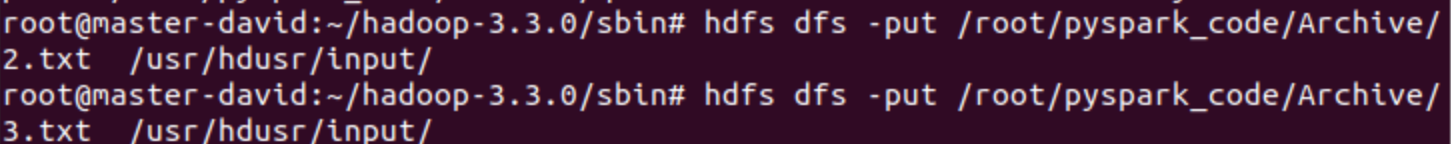
Firstly, start the hdfs and yarn by the command: start-all.sh

Or just follow the command start-yarn.sh and start-dfs.sh

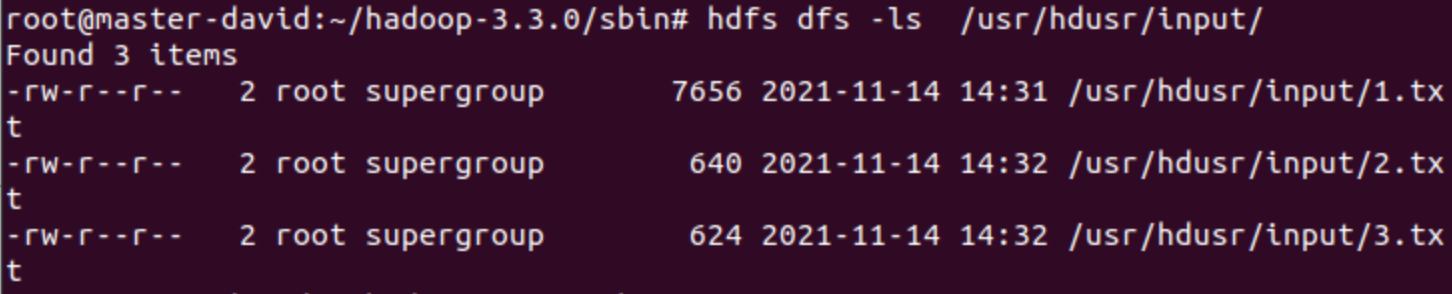


Then put the 1.txt, 2.txt, 3.txt three documents to the hdfs.

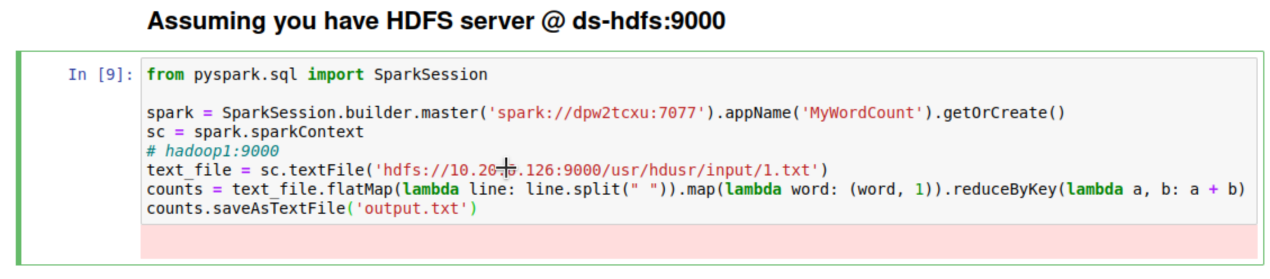




Then use this command - to check the text file whether upload sucessfully.

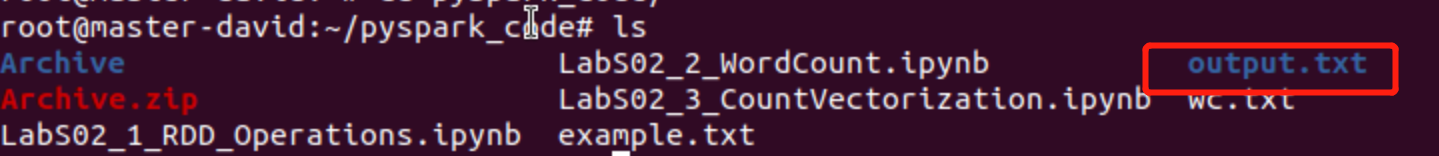


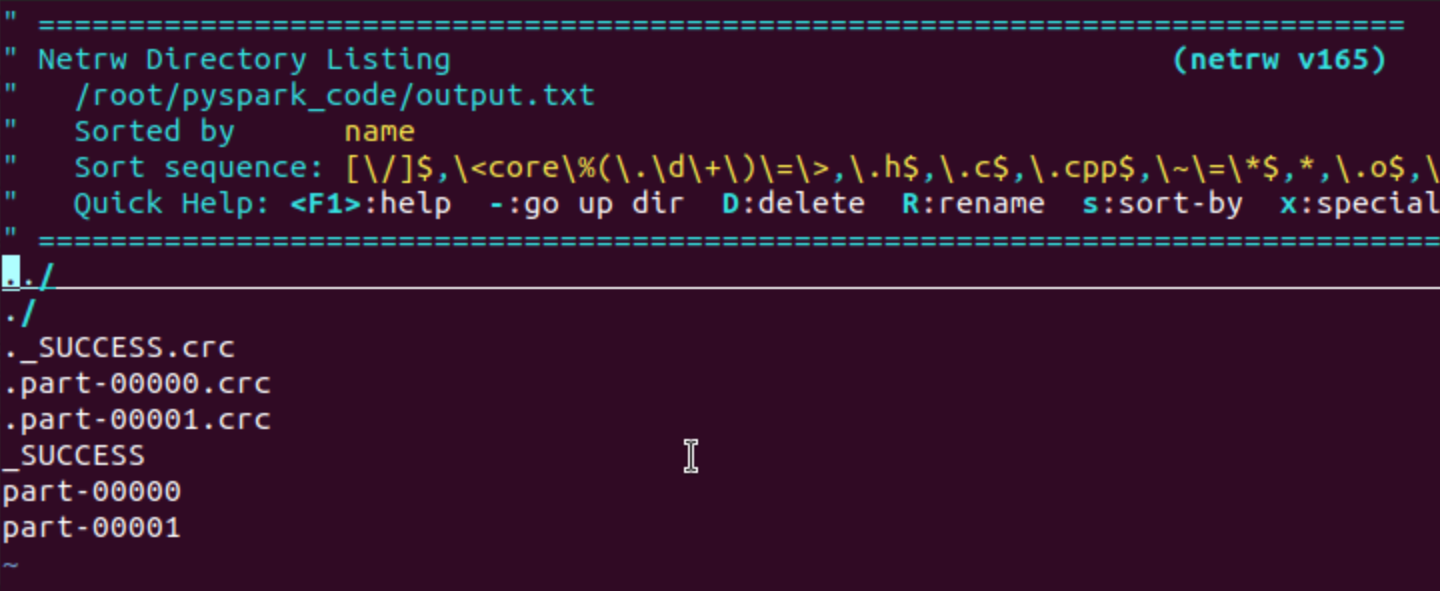
Change the ds-hdfs to your master host name



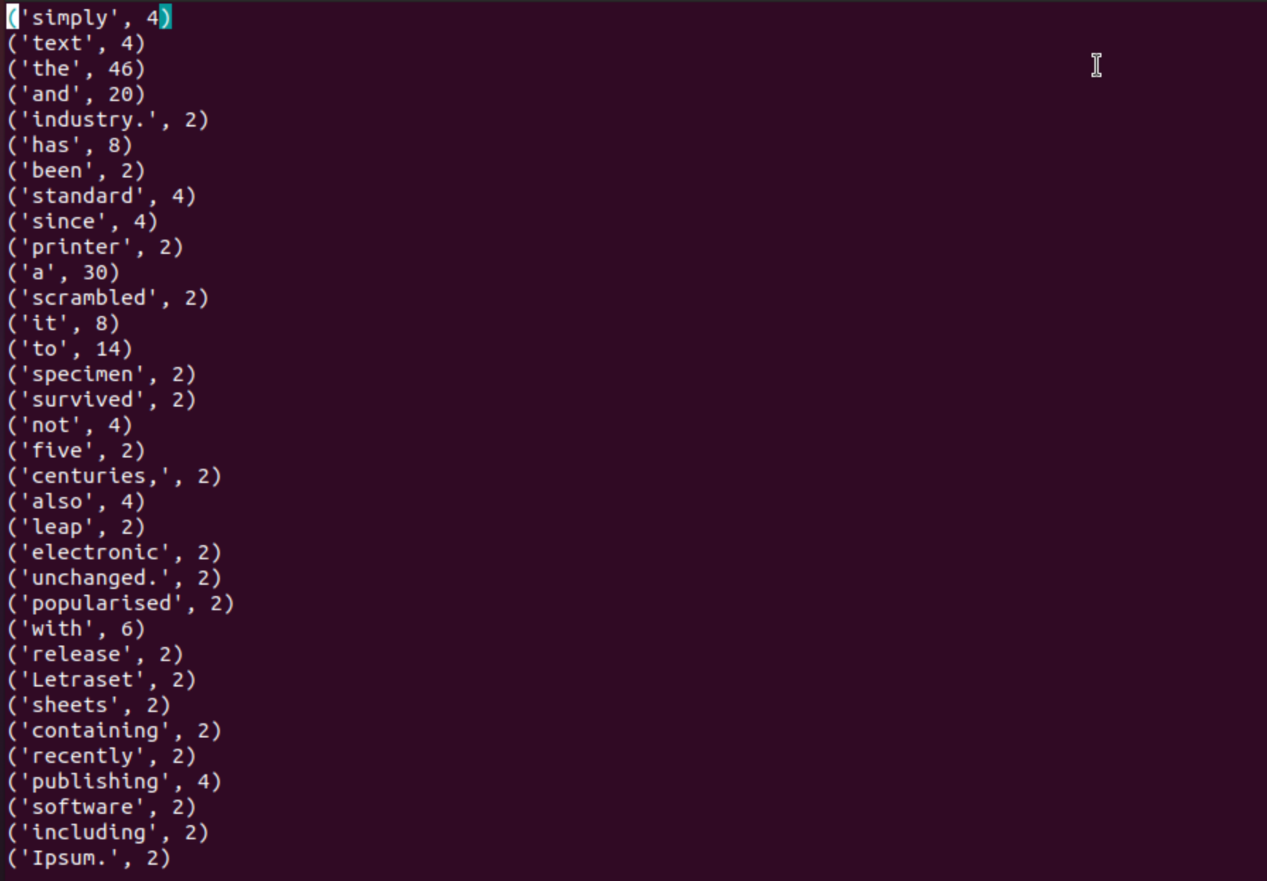
And you can change the textFile parameter to the path you upload the threee document just now. **Pay attention to the url is hdfs:// + “your slaves private ip” + “path”**

After it finish runing, check the output file.





-

**

If you can see this result, that means you are sucessful.

The third project:|

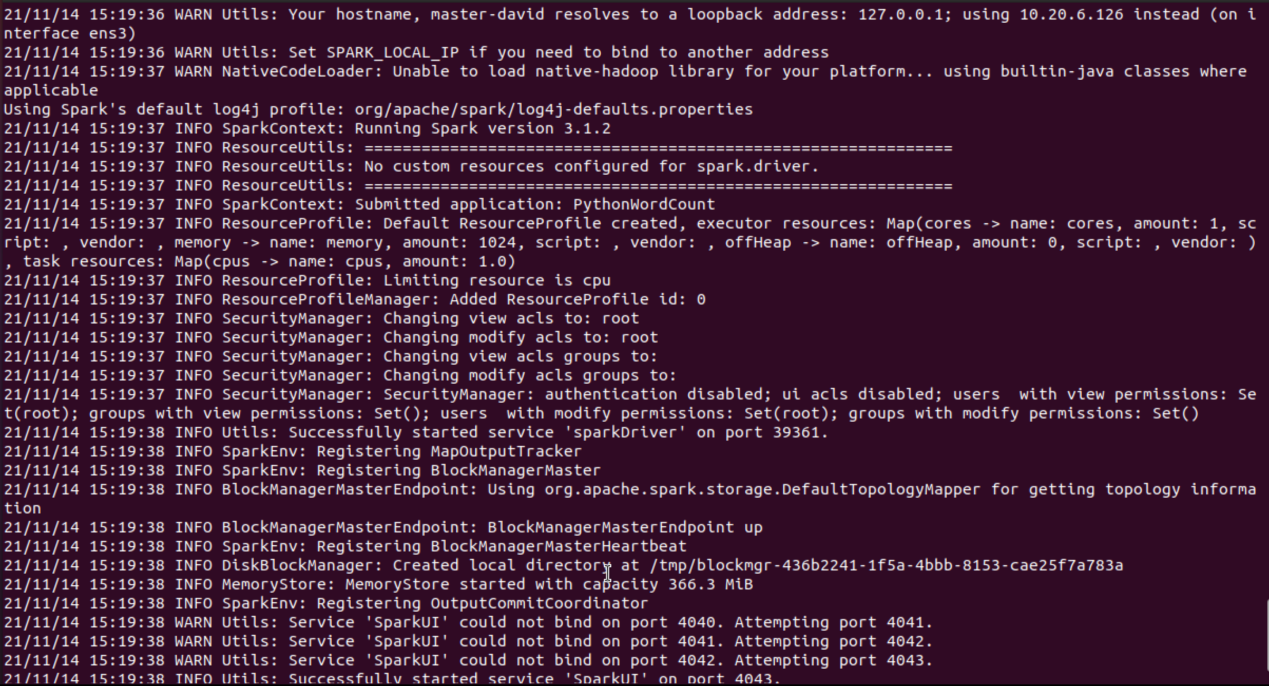
You can also run this command from CLI of your Spark Master node:

Firstly, get into your spark basic home by the command：

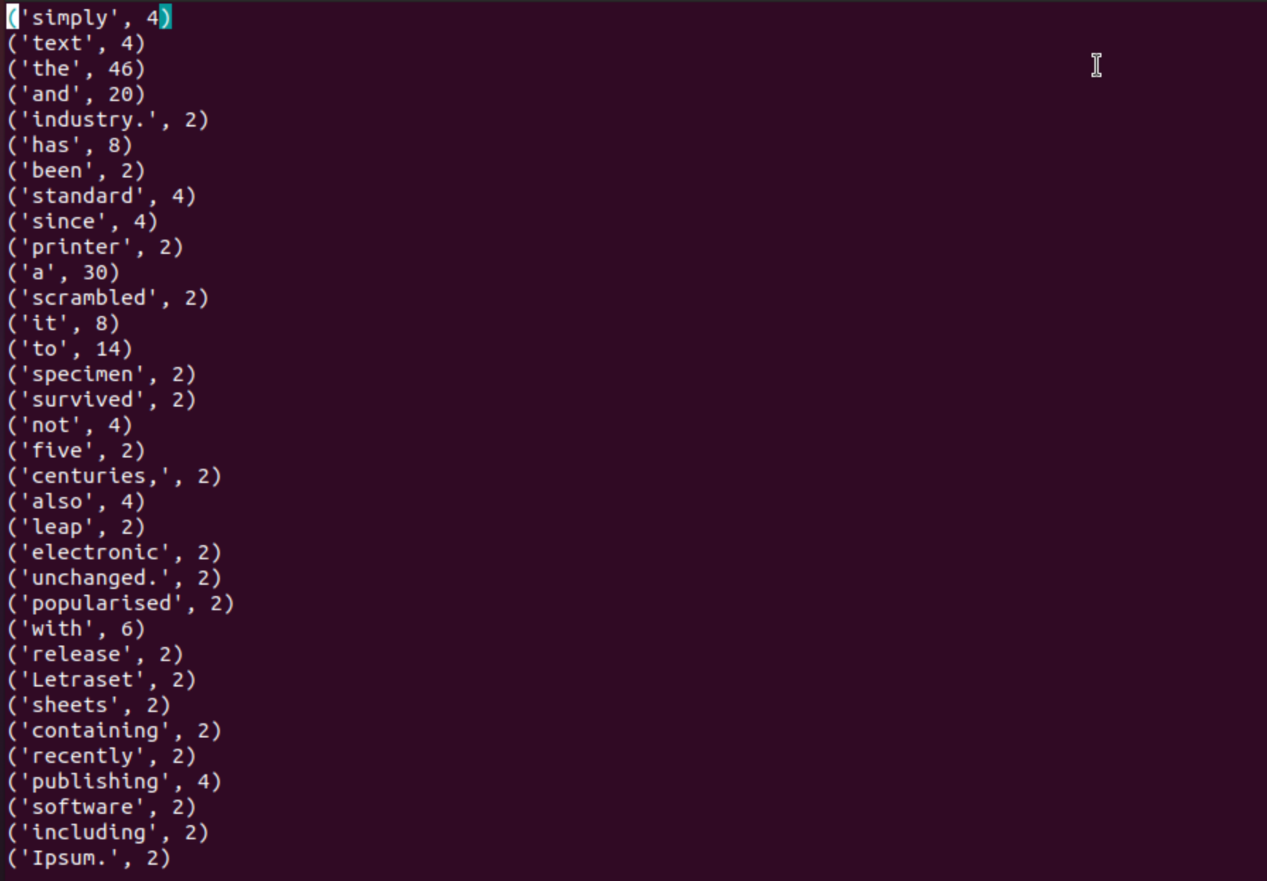
Then excute the command at this directory:

--

(The dfs path is same as the second project and “master” is your master\_namenode’s host name)



Successfully started service.

**

End!