WordCount in Spark

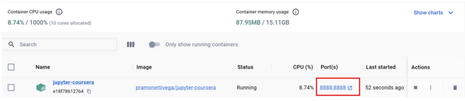
By the end of this activity, you will be able to:

1. Perform WordCount with Spark Python (PySpark)

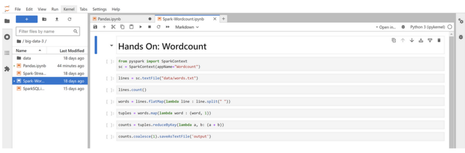
For this activity, you should have completed the creation of the JupyterLab container. If not follow, Steps 1-3 on the previous activity *Hand On: Exploring Pandas DataFrames*, and then come back to Step 2 of this activity.

Step 1. Start the container. Open Docker Desktop and start your *jupyter-coursera* container.

When Jupyter starts running, click on the port to access JupyterLab in your browser:



Step 2. Open your notebook. Once you're in JupyterLab, go to the *big-data-3* folder and open the *Spark-Wordcount.ipynb* notebook.



Step 3. Start your SparkContext. To establish a connection with Spark, the first thing we do is to start a SparkContext.

```
[1]: from pyspark import SparkContext
     sc = SparkContext(appName="Wordcount")
```

Step 4. Read Shakespeare. For this activity, we are going to use a words.txt text file, which contains the works of William Shakespeare. We can read the text and create a new variable called *lines:*

```
[2]: lines = sc.textFile("data/words.txt")
```

The *textFile()* method reads the file into a Resilient Distributed Dataset (RDD) with each line in the file being an element in the RDD collection.

We can verify the file was successfully loaded by calling the *count()* method, which prints the number of elements in the RDD:

```
[3]: lines.count()

[3]: 124456
```

Step 5. **Split each line into words**. Next, we will split each line into a set of words. To split each line into words and store them in an RDD called *words*, run:

```
[4]: words = lines.flatMap(lambda line : line.split(" "))
```

The *flatMap()* method iterates over every line in the RDD, and *lambda line : line.split(" ")* is executed on each line. The *lambda* notation is an anonymous function in Python, i.e., a function defined without using a name. In this case, the anonymous function takes a single argument, *line*, and calls *split(" ")* which splits the line into an array words.

Step 6. **Assign initial count value to each word**. Next, we will create tuples for each word with an initial count of 1:

```
[5]: tuples = words.map(lambda word : (word, 1))
```

The *map()* method iterates over every wordin the *words* RDD, and the *lambda* expression creates a tuple with the word and a value of 1.

Note that in the previous step we used *flatMap*, but here we used *map*. In this step, we want to create a tuple for every word, i.e., we have a one-to-one mapping between the input words and output tuples. In the previous step, we wanted to split each line into a set of words, i.e., there is a one-to-many mapping between input lines and output words. In general, use *map* when the number of inputs to number of outputs is one-to-one, and *flatMap* for one-to-many (or one-to-none).

Step 7. Sum all word count values. We can sum all the counts in the tuples for each word into a new RDD *counts*:

```
[6]: counts = tuples.reduceByKey(lambda a, b: (a + b))
```

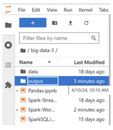
The *reduceByKey()* method calls the *lambda* expression for all the tuples with the same word. The lambda expression has two arguments, *a* and *b,* which are the count values in two tuples.

Step 8. Write word counts to text file. We can write the *counts* RDD to a text file:

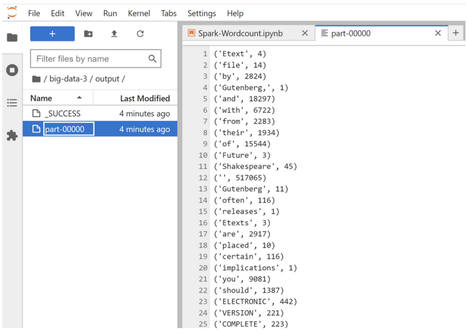
```
[7]: counts.coalesce(1).saveAsTextFile('output')
```

The *coalesce()* method combines all the RDD partitions into a single partition since we want a single output file, and *saveAsTextFile()* writes the RDD to the specified location.

Step 9. View results. We can view the results by opening the new *output* directory:



And then opening the *part-00000* file:



Step 10. Exiting the container. To exit JupyterLab, simply close the tab in your browser. To stop the container, go to Docker Desktop and click on the *stop* button. We recommend not to delete the container, as this container will be used for multiple activities across this specialization.

Go to next item

Completed

Like Dislike Report an issue