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

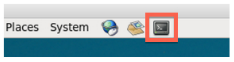
1. Read and write text files to HDFS with Spark
2. Perform WordCount with Spark Python

This activity requires programming in a Jupyter Notebook. If you have not already started the Jupyter Server, do that now. The instructions for starting the server can be found in the Reading *Instructions for Starting Jupyter Server* at the end of Week 1.

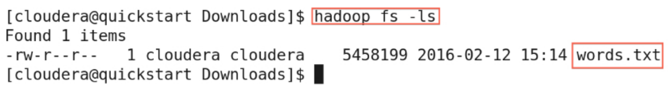
[NOTE: if you would like to copy and paste commands into your terminal window, you can access the completed Python Notebook for this reading at **https://github.com/words-sdsc/coursera/blob/master/big-data-3/notebooks/Spark%20WordCount.ipynb**]

Step 1. **Check if Shakespeare text is in HDFS**. In this activity, we will count the words in the complete works of Shakespeare. In Course 1: Intro to Big Data, we downloaded and copied a text file containing these works to HDFS. Let's check to see if the file is still in HDFS. If the file is not there (or you did not take Course 1), there are instructions below for copying the file into HDFS.

Open a terminal window by clicking on the square black box on the top left of the screen.



We can see the contents of HDFS by running *hadoop fs -ls*:



```
[cloudera@quickstart Downloads]$ hadoop fs -ls
Found 1 items
-rw-r--r--   1 cloudera cloudera    5458199 2016-02-12 15:14 words.txt
[cloudera@quickstart Downloads]$
```

In Course 1, we saved the complete works of Shakespeare as *words.txt*. If you see this file in HDFS, proceed to Step 2. If this file is not in HDFS, let's copy it. Change directory to *Downloads/big-data-3/spark-wordcount*.

Copy the file to HDFS:

hadoop fs -put words.txt

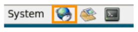
If you re-run *hadoop fs ls*, you should see words.txt.

TO execute a Line of code in the Jupyter notebook, either:

a). Click on the play next button as you would on a DVD player

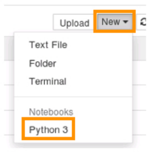
b). Hold down Shift and press Enter

Step 2. **Open a web browser and create a new Jupyter Python Notebook.** Open a web browser by clicking on the web browser icon at the top of the toolbar:

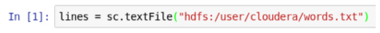


Navigate to *localhost:8889/tree/Downloads/big-data-3*:

Create a new Python Notebook by clicking on *New*, and then click on *Python 3:*



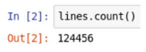
Step 3. **Read Shakespeare text from HDFS**. We can read the text in HDFS into a new variable called *lines:*



```
In [1]: lines = sc.textFile("hdfs:/user/cloudera/words.txt")
```

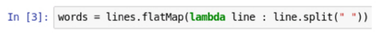
The SparkContext, *sc*, is the main entry point for accessing Spark in Python. The *textFile()* method reads the file into a Resilient Distributed Dataset (RDD) with each line in the file being an element in the RDD collection. The URL *hdfs:/user/cloudera/words.txt* specifies the location of the file in HDFS.

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



```
In [2]: lines.count()
Out[2]: 124456
```

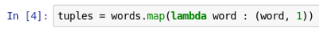
Step 4. **Split each line into words**. Next, we will split each line into a set of words. To split each line into words and store them in an RDD called *words*, run:



```
In [3]: 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 5. **Assign initial count value to each word**. Next, we will create tuples for each word with an initial count of 1:



```
In [4]: tuples = words.map(lambda word : (word, 1))
```

The *map()* method iterates over every word in 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).

number of inputs to number of outputs is one-to-one, and *flatMap* for one-to-many (or one-to-none).

Step 6. **Sum all word count values**. We can sum all the counts in the tuples for each word into a new RDD *counts*:

```
In [5]: 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 7. **Write word counts to text file in HDFS**. We can write the *counts* RDD to HDFS:

```
In [6]: counts.coalesce(1).saveAsTextFile('hdfs:/user/cloudera/wordcount/outputDir')
```

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 8. **View results**. We can view the results by first copying the file from HDFS to the local file system and then running *more*:

```
hadoop fs -copyToLocal wordcount/outputDir/part-00000 count.txt
more count.txt
```

Mark as completed