**Exercise 7**

*Get started with Apache Spark and Python*

**Prior Knowledge**

Unix Command Line Shell

Simple Python

**Learning Objectives**

Understand the Spark system

Understand the Jupyter Notebook model

WordCount!

**Software Requirements**

(see separate document for installation of these)

* Apache Spark 3.0.0
* Python 3.8.x
* Jupyter notebooks

**Part A. Spark Python Shell (pySpark)**

1. We are going to do a wordcount against a set of books downloaded from Project Gutenberg. Wordcount is the definitive Big Data program (sort of Hello World for Big Data) and it is frankly embarrassing that we haven’t done one yet.
2. Apache Spark has a useful Python shell, which we can use to interactively test and run code.
3. We are going to start by using data in HDFS, so *we need to ensure HDFS is running.* (Follow the instructions from the Hadoop lab).
4. Let’s load some books into HDFS. In a terminal window (Ctrl-Alt-T)

hadoop fs -mkdir -p /user/oxclo/books

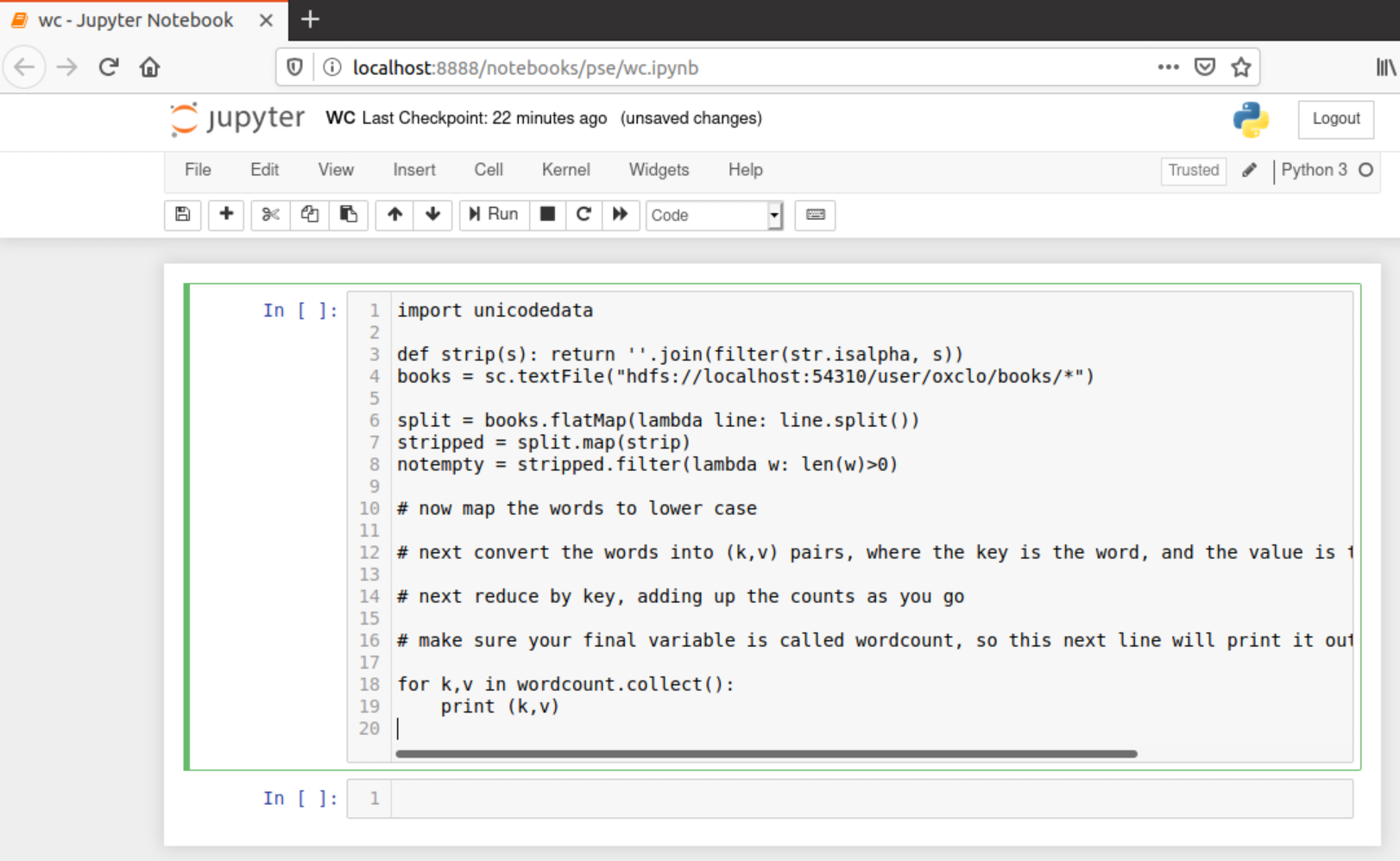
hadoop fs -put ~/datafiles/books/\* /user/oxclo/books/

1. Let’s make a directory for our code:  
     
   mkdir ~/pse  
   cd ~/pse
2. Now start the Spark Python command line tool –  
     
   pyspark
3. In the command-line you will see something like
4. Jupyter is starting. This is because I have preconfigured the following environment variables in ~/.bashrc

export PYSPARK\_DRIVER\_PYTHON=jupyter

export PYSPARK\_DRIVER\_PYTHON\_OPTS='notebook'

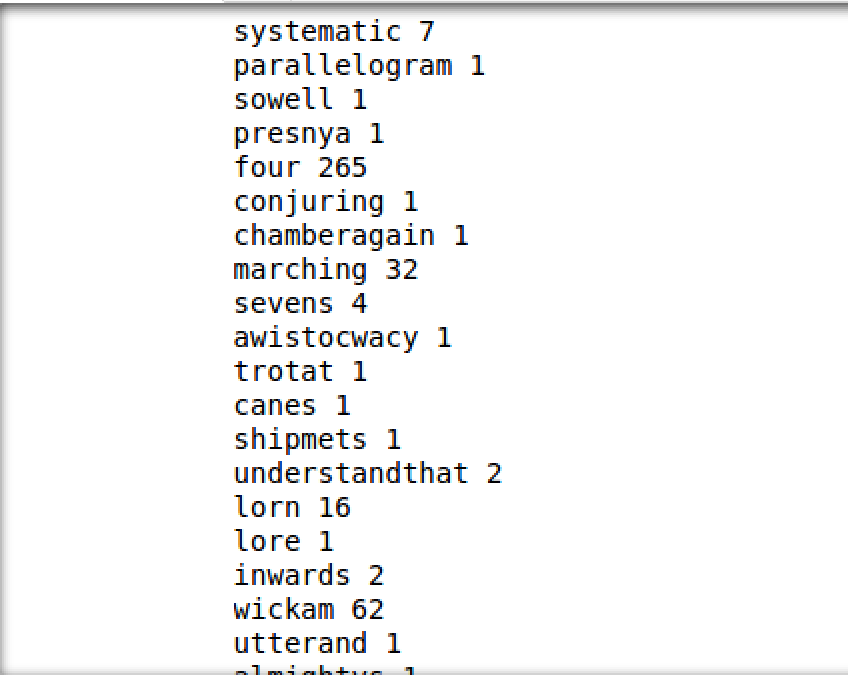
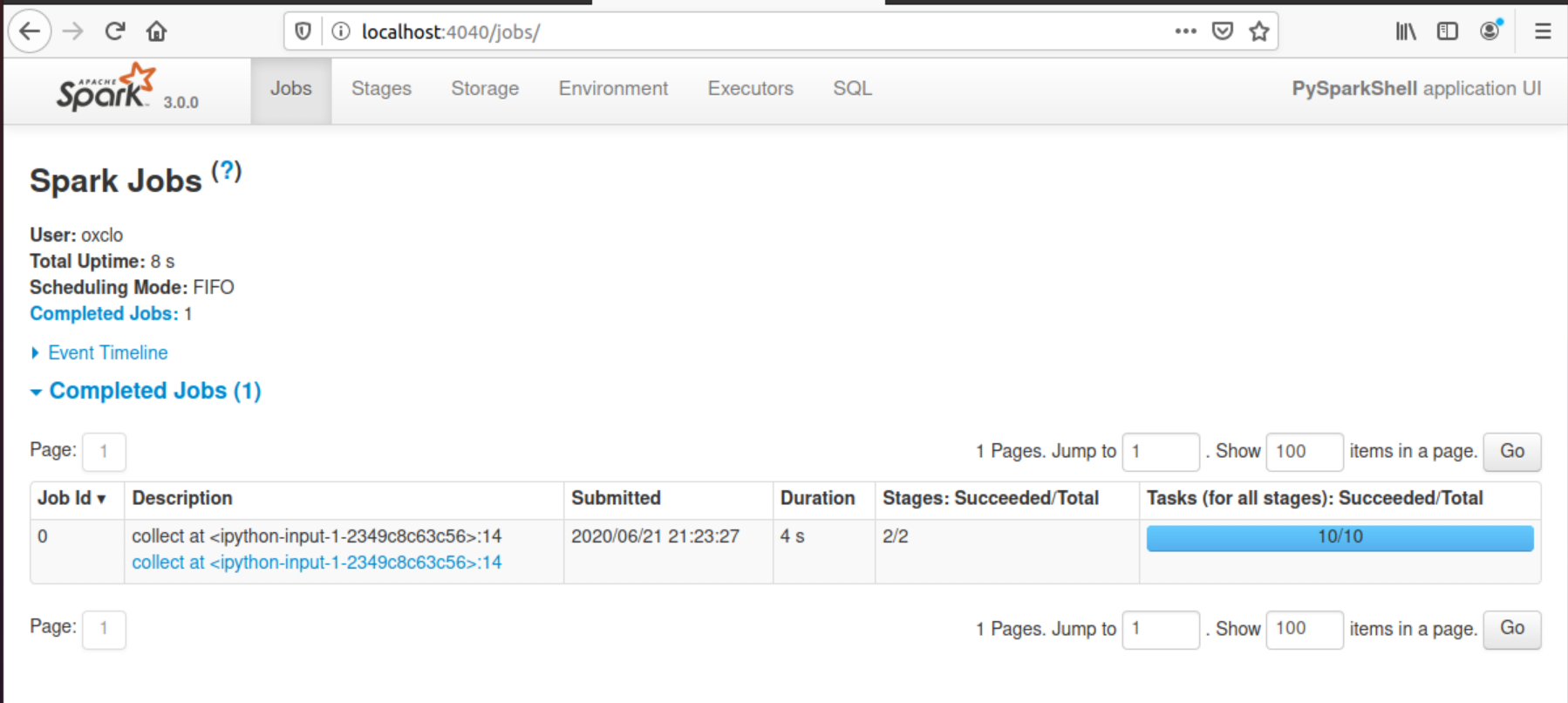
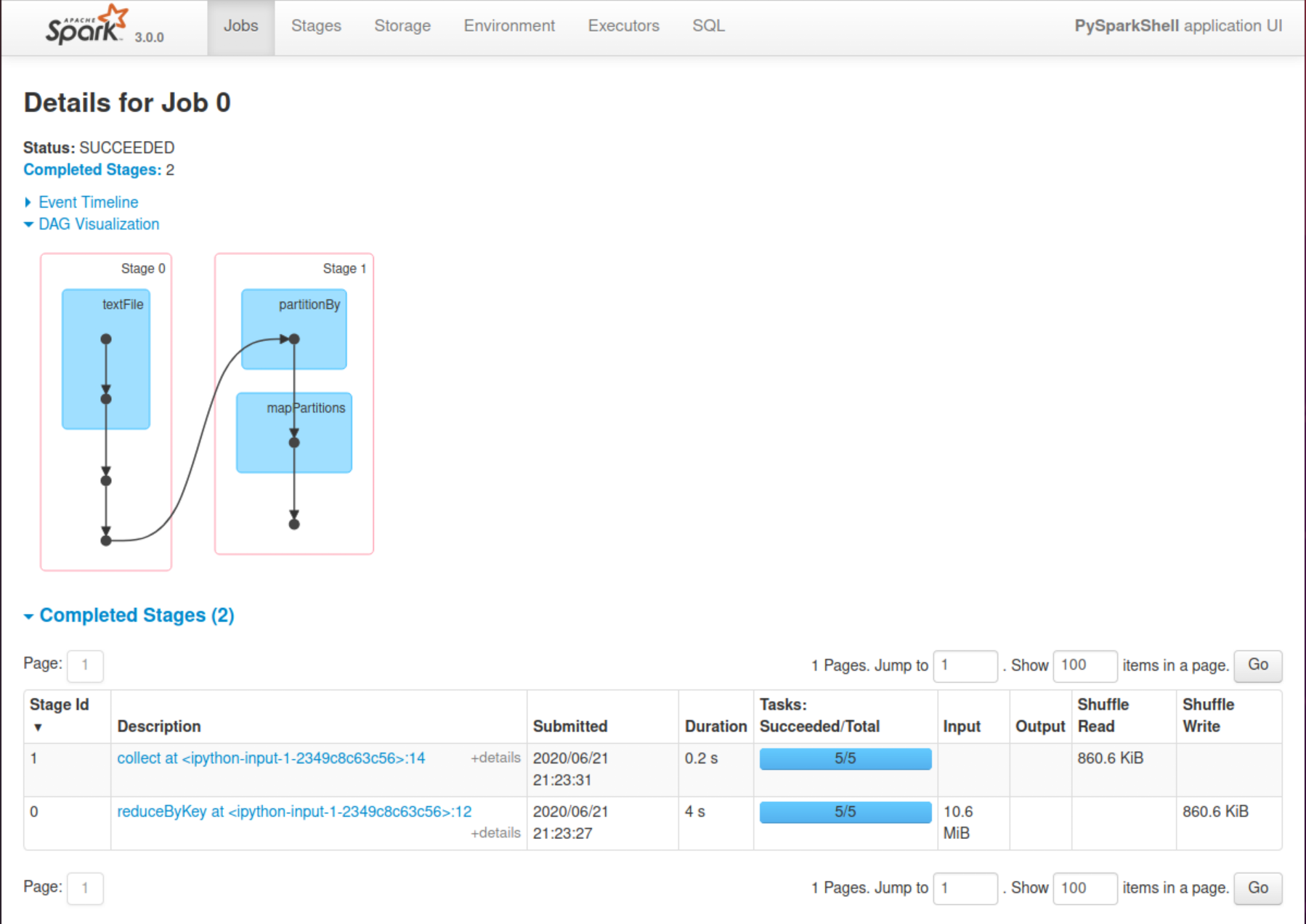
If you want to use the more basic version of pyspark, you’ll have to unset these.

1. And then a browser window will pop up.  
   
2. Use the **New** button to create a new Python3 notebook:  
   
3. There is a starter of the code you need in the following URL:   
   <https://freo.me/first-notebook>   
     
   Paste that into the cell [1] so it looks like this:
4. There are some aspects that are not filled in that you need to write.   
   Basically this is a data-processing pipeline (also a directed acyclic graph)
5. *Let’s look at the parts that are there already.*
6. We already have a SparkContext object defined in the notebook (in a program you need to define one, which we will see later)
7. We want to remove any non-alphanumeric characters. This is a function that will do that:  
     
   def strip(s): return ''.join(filter(str.isalpha, s))
8. With the preliminaries over, the next line loads the data in:

books = sc.textFile("hdfs://localhost:54310/user/oxclo/books/\*")

1. Then splits the lines into separate words  
     
   split = books.flatMap(lambda line: line.split())
2. Removes non-alpha characters:  
     
   stripped = asc.map(strip)  
     
   and removes empty items:

notempty = stripped.filter(lambda w: len(w)>0)

1. Now it is time for you to do something!  
     
   Convert all the words to lower case, using a map operation. In python, if *str* is a string, then str.lower() is the same string in lower case.
2. Now you need to get ready for a reduce. In order to do a reduce, we need some form of *key, value* pairs. I recommend using *tuples*  which are simply (k,v) in Python (the brackets group the items into a tuple).
3. Remembering how reduce works, we need each word to have a count. Before reducing, that count is 1. So we need a lambda that takes a word w and returns (w,1)
4. Now we can do a reduce that adds all those counts together.
5. Finally, we need to collect the results and print them. In Spark, they may be distributed across different RDD partitions on different machines, so the collect() method brings them together.  
     
   for k,v in wordcount.collect():   
    print (k,v)
6. Try running the cell, by clicking 
7. Be patient. I suggest you look at the command window and wait until you see spark start working.
8. You should see a word count appear below cell 1:
9. Congratulations!
10. While the pyspark is still running browse to <http://localhost:4040>
11. You will see the Spark web console:  
    
12. Click on the blue link “collect at ipython-input”  
    This shows you how Spark converted your code into stages:  
    
13. Click on Stage 0  
    
14. And expand the Event Timeline:
15. Make sure your code is saved from the notebook.
16. Quit the notebook shell by typing Ctrl-C on the command line, and then Y  
    Also close the notebook windows in the browser.
17. Now let’s run the same code as a “job” instead of interactively.
18. From <http://freo.me/oxclo-wc-py> copy the code into a file wc-job.py
19. You will notice that there is a bunch of “setup” code that we didn’t need in the pyspark command line tool. That is because pyspark assumes you want all this and does it for you.
20. The default environment in your terminal session is telling Pyspark to use Jupyter. We need to stop that. Type:

unset PYSPARK\_DRIVER\_PYTHON

unset PYSPARK\_DRIVER\_PYTHON\_OPTS

Note that this will reset when you start a new terminal window, so when you want to go back to using Jupyter, just kill this window and start another.

1. We run jobs locally on a single node directly on Spark:  
   The local[\*] indicates to use as many threads as you have cores on your system:  
   ~/spark/bin/spark-submit --master local[\*] wc-job.py "hdfs://localhost:54310/user/oxclo/books/\*"
2. Congratulations, the lab is complete!  
     
   **Extension**
3. Re load the code into the Jupyter notebook and now improve it to show the wordcount in descending order, starting with the most common words.