How to install Spark on Windows (optional)

You can refer to this useful guide
http://www.ics.uci.edu/~shantas/Install_Spark_on_Windows10.pdf

How to install Spark on your Linux/Unix machine (optional)

Go to this web page https://spark.apache.org/downloads.html and select version 2.3.0 of Spark (or choose the latest version).

Select the 'Pre-built for Hadoop 2.7' version, then download the file spark-2.3.0-bin-hadoop2.7.tgz into a directory of your choice.

By using a terminal, go into that directory and use the following command to unzip the file:
tar xzvf spark-2.3.0-bin-hadoop2.7.tgz

You will see that a directory spark-2.3.0-bin-hadoop2.7 will appear in the same directory.
Now you just need the following command to launch **pyspark:** spark-2.3.0-bin-hadoop2.7/bin/pyspark

The spark 2.x.x run on java 8, therefore, if your java version is higher than 8, and Java 8 is available on your machine you can run pyspark with the command below:
cd spark-2.3.0-bin-hadoop2.7
JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64/ bin/pyspark

At this point you can use HDFS (see next step) and run MapReduce jobs coded in Python, for instance the WordCount job as indicated in the following.

Your login user name is 'hadoop' and in HDFS your home is at this path: **/user/hadoop.**
Useful commands for Linux and HDFS. See the following documents that we borrowed from the Web:
https://www.dropbox.com/s/77z7m35tfe18jyr/hadoop-hdfs-commands-cheatsheet.pdf
https://www.dropbox.com/s/06qw849h2omjke0/fwunixref.pdf

Using the Python shell in Spark on the cluster

Once you are connected to the cluster, just use this command
> pyspark
In order to activate the shell and interact with Spark in Python.
Note: To active the "*Tab Autocomplete*" and "*print*" in pyspark shell, use the below scripts:

- import rlcompleter, readline
- readline.parse_and_bind("tab: complete")
- from __future__ import print_function

Data Science for Business (Lab session)

Exercise 1.

Encoding SQL queries in MapReduce. Consider the following simple relational schema containing information's about clients and orders they made.

For testing the jobs use the following files (use wget *file-url* for downloading them)

https://www.dropbox.com/s/tmt6u80mkrwfjkv/Customer.txt

https://www.dropbox.com/s/8n5cbmufqhzs4r3/Order.txt

Customer(#cid, startDate, name)

Order(#cid, total)

Note that, for the sake of simplicity, we do not have any primary key for Order. Also assume that all the fields are mandatory.

In order to create the initial RDD, save your text file on HDFS (if HDFS installed on the machine) or local file system and then use the following command to read the file and transform it into an RDD which we name 'document'.

>>> document = sc.textFile("hdfs://…path to … your .txt" document)
OR
>>> document = sc.textFile("file:///home/…path to … your .txt" document)

Observe that sc is a predefined object (sc stands for Spark Context) automatically created by the pyspark shell. sc is used as a gateway to the Spark world.

Provide the Python a Spark algorithm of the following SQL queries.

- SELECT name FROM Customer WHERE month(startDate)=7
- SELECT DISTINCT name FROM Customer WHERE month(startDate)=7
- SELECT O.cid, SUM(total), COUNT(DISTINCT total) FROM Order O GROUP BY O.cid
- SELECT C.cid, O.total FROM Customer C, Order O WHERE month(startDate)=7 and C.cid=O.cid

Data Science for Business (Lab session)

Exercise 2.

Consider a text file of your choice or link below, and design and implement in pySpark an algorithm for the word-count problem (How many unique words available in the text?).

https://www.dropbox.com/s/2f3nt4rn7t4wee1/5000-8.txt

Exercise 2.1.

Consider the previous document when the data separated by white space from each other. Design an algorithm for pyspark to counting unique words per line.

Data Science for Business (Lab session)

Exercise 3.

Consider the below pets RDD and design a Spark script for compute the average quantity of each pet.

pets = sc.parallelize([("cat", 1), ("dog", 1), ("cat", 2),("dog", 3)])

Note: Still use pyspark.

Data Science for Business (Lab session)

Exercise 4.

Imagine two text files randomly generate called F1.txt and F2.txt.

F1.txt including points/vectors in R^5. For instance, a point in the files is represented as: '3.45, 5.0, 0, 0, 2.0'.

Consider that F1.txt should contain at least 5 thousand points, while F2.txt must contain exactly 10 points.
The output of the jobs can be concatenated in order to build Fi.txt.

Questions:

1. Do you really need a Spark job to generate F2.txt?
2. Also, how you could ensure that no two tasks generate the same point?

Exercise 5.

Consider files F1.txt and F2.txt are on HDFS (or local file), inside a directory that you name InputForCrossProduct, under /user/hadoop (or /home/...).

Design a *pyspark* job that takes as input the two files and builds the cross-product of F1.txt and F2.txt.
To illustrate if F1.txt contains '3.45, 5.0, 0, 0, 2.0' and F2.txt contains '2.45, 1.0, 2.0, 2.0', then the output of the job must contain their concatenation

'3.45, 5.0, 0, 0, 2.0 2.45, 1.0, 2.0, 2.0'

where '\t' is used to separate the two vectors.

If needed, do not hesitate to pre-process F2.txt, even on the local file system. Also assume that 10, the cardinality of F2.txt, is initially known by the job you design.

Exercise 6.

Solve exercises 4 and 5 with **Spark**. You are not allowed to use the Spark cross product transformation. You can use the **zipWithIndex** transformation operation to add a unique index to each element of an RDD. For instance:

```
>>> rdd1 = sc.parallelize(['a','d','x'])
>>> rddzipped = rdd1.zipWithIndex()
>>> rddzipped.collect()
[('a', 0), ('d', 1), ('x', 2)]
```

Exercise 7:
Do you think it is possible to perform an equi-join between two RDDs without using the join() transformation. If so, find the algorithm and code it into Spark. Just use the key-value RDDs used in classes to test your code.

Does the same holds for **right-outer** join? If so, proceed as before.

What is your solution for the inner join?

Data Science for Business (Lab session)

Exercise 8.

Graph analytics. Design and implement algorithms in Spark for the following analytics problems on directed graphs. Assume that a graph is a set of edge pairs (v, w) indicating that an edge exists from node v to node w.

Sinks. Given a directed graph, find the set of nodes having only incoming edges and having no outgoing edges.

Universal sinks. Given a directed graph, find the set of nodes having an incoming edge from all the remaining nodes, and having no outgoing edges.

You can use *sc.parallelize* to create a small RDD containing pairs for a graph to use as input for your algorithms.

Assume the graph is stored in the following file.
https://www.dropbox.com/s/3bre2e1jbsysxej/graph.txt

Note: Please consider that each line is in the form of « v edge-label w w », so in order to represent a couple « v w » only the first and third elements should be retained for the sink calculation.