Data Science and AI for industrial systems

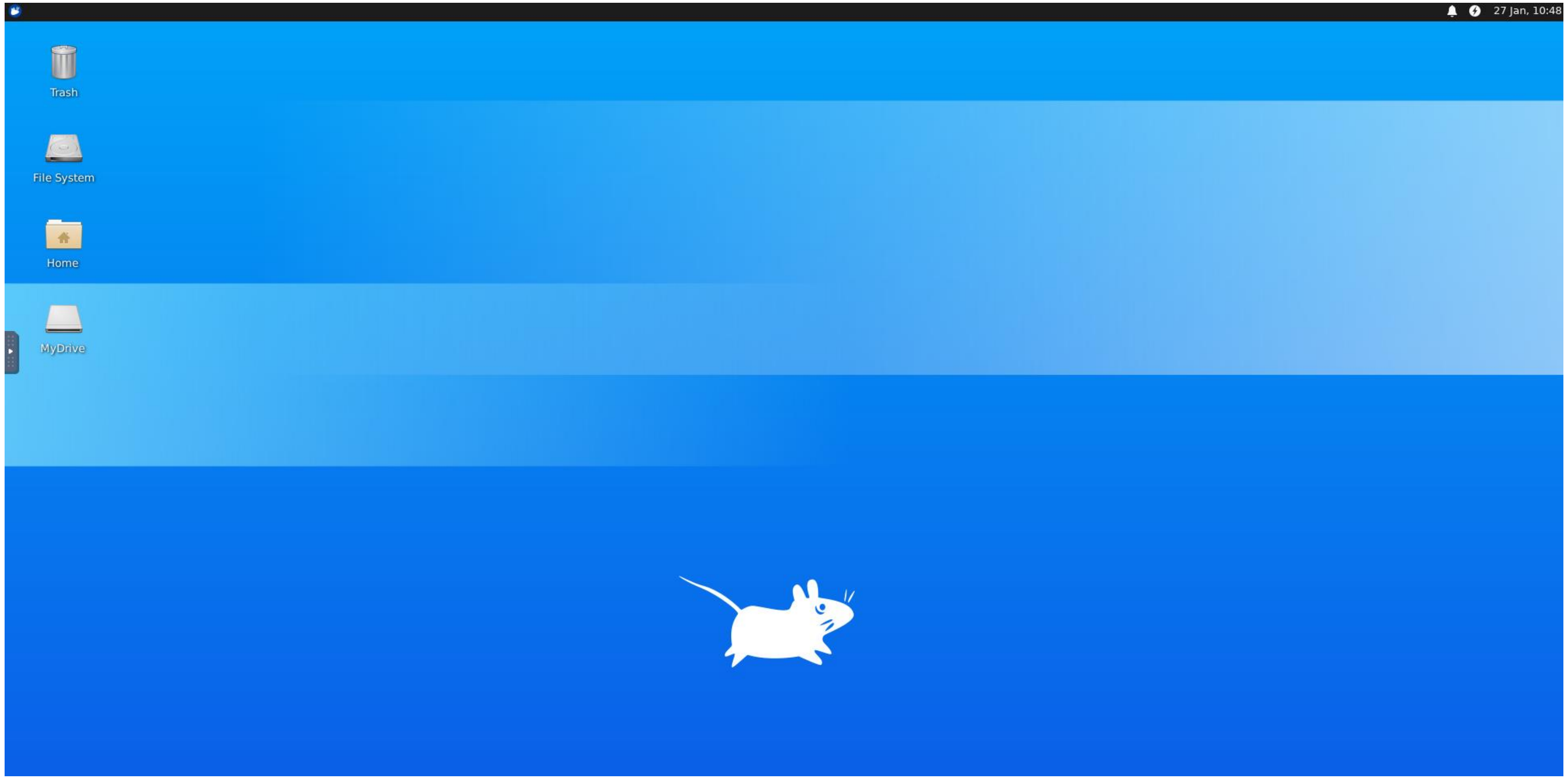
Hands-on session 1
Intro to PySpark

Prof. Daniele Apiletti, Simone Monaco

03/02/2023

Here we are - CrownLabs

```
https://crownlabs.polito.it
```

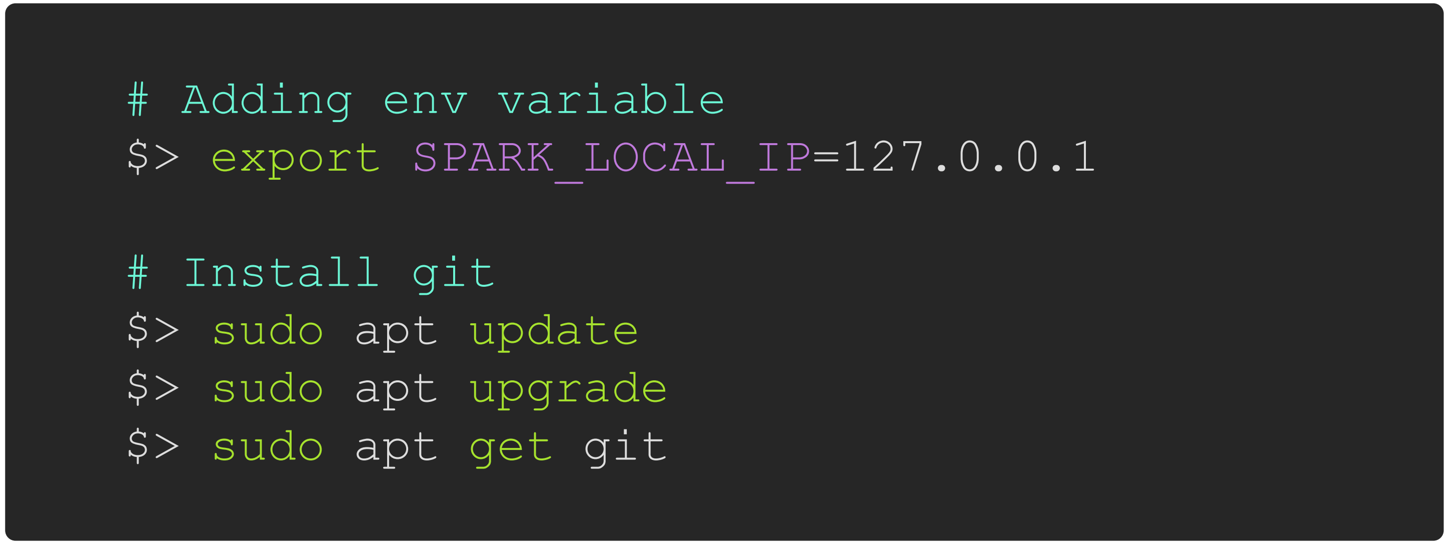


Configuration

We need to run locally **PySpark** applications on a **Jupyter Notebook,** to do so, let's:

- Add useful **environment variables**
- Install git to get access to the material

Configuration



```
# Adding env variable
$> export SPARK_LOCAL_IP=127.0.0.1

# Install git
$> sudo apt update
$> sudo apt upgrade
$> sudo apt get git
```

Lab material

You will find all the material (slides, text and solutions) in the following repository:

https://github.com/dbdmg/aiis-mllabs

Or better (https://bit.ly/aiislabs)

To get access to it

```
$> git clone https://github.com/dbdmg/aiis-mllabs.git
```

Any time you want to update

```
$> git pull
```

Theory update - PairRDD

- What is the number of people per nation?

It would be useful to perform operations separately for each nation (the **key** of the sample) separately from the rest (the **value** part).

```
namesPairRDD = namesRDD\
       .map(lambda x: (x.split(",")[1], 1))
```

```
> [("Morocco", 1),

   ("Italy", 1),

   ("England", 1)

  …]
```

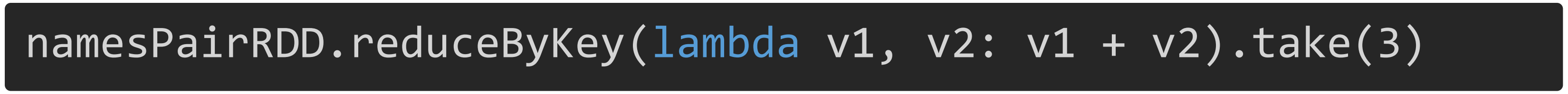
Input file

Abdul,Morocco
Mario,Italy
Chloe,England
Henry,England
Carmen,Spain
Xiu,China
Giovanna,Italy
Bernadette,France
…

Pair Actions

- RDDs of key-value pairs are characterized by the operations available for the “standard” RDDs
    - **`filter()`** , **`map()`** , **`reduce()`**, etc.

- BUT they are also characterized by specific operations
    - **`reduceByKey()`**, **`join()`**, etc.
    - These operations analyses the content of one group (key) at a time



```
namesPairRDD.reduceByKey(lambda v1, v2: v1 + v2).take(3)
```

```
> [("Morocco", 10),
   ("Italy", 25),
   (“England", 7)]
```

pairRDD with flatMap transformation

- Define an RDD of key-value pairs by using the `flatMap(f)` transformation
- Apply a function **f** on each element of the input RDD that returns a list of tuples for each input element
    - The new PairRDD contains all the pairs obtained by applying **f** on each element **x** of the “input” RDD
        - **[y]= f(x)**
            - Given an element **x** of the input RDD, **f** applied on **x** returns a list of pairs **[y]**
            - The new RDD is a “list” of pairs contains all the pairs of the returned list of pairs. It is NOT an RDD of lists.
        - **[y]** can be the empty list

Example: Word count

- Create an RDD from a textual file, each line of the file contains a set of words

Input file:

```
Lorem ipsum sit\n
amen dolor…
```

```
# Define f
def wordsOnes(line):
   pairs = []
for word in line.split(' '):
   pairs.append( (word, 1) )
   return pairs

linesRDD = sc.textFile("document.txt")

# Create an RDD of key-value pairs
# One pair (word,1) for each input word
wordOnePairRDD= linesRDD.flatMap(wordsOnes)
```

```
["Lorem ipsum sit",
"amen dolor"…]
```

```
1: [(Lorem,1), (ipsum ,1),(sit,1)], …
```

```
2: […,
    (Lorem,1),
    (ipsum ,1),
    (sit,1),
    …]
```

Useful transformations

- **`reduceByKey(f)`**: Create a new RDD of key-value pairs with one pair **for each** distinct key k of the input RDD of key-value pairs, applying a function f on the values v.
- **`groupyKey()`**: Create a new RDD of key-value pairs with one pair **for each** distinct key k, with as value the list of values associated with k in the input
- **`mapValues(f)`**: A `map(f)` transformation applied only on the value part of each pair
- **`keys()/values()`**: Return an RDD with the key/values part only
- **`join(otherRDD)`**: Join the key-value pairs of two RDDs of key-value pairs based on the value of the key of the pairs
  - the result for each key common to both has the form `(key, (value1, value2))`

Let's move to the code