Data Science and AI for industrial systems

Hands-on session 1.1
Intro to PySpark

Prof. Daniele Apiletti, Simone Monaco

PySpark

17/02/2023

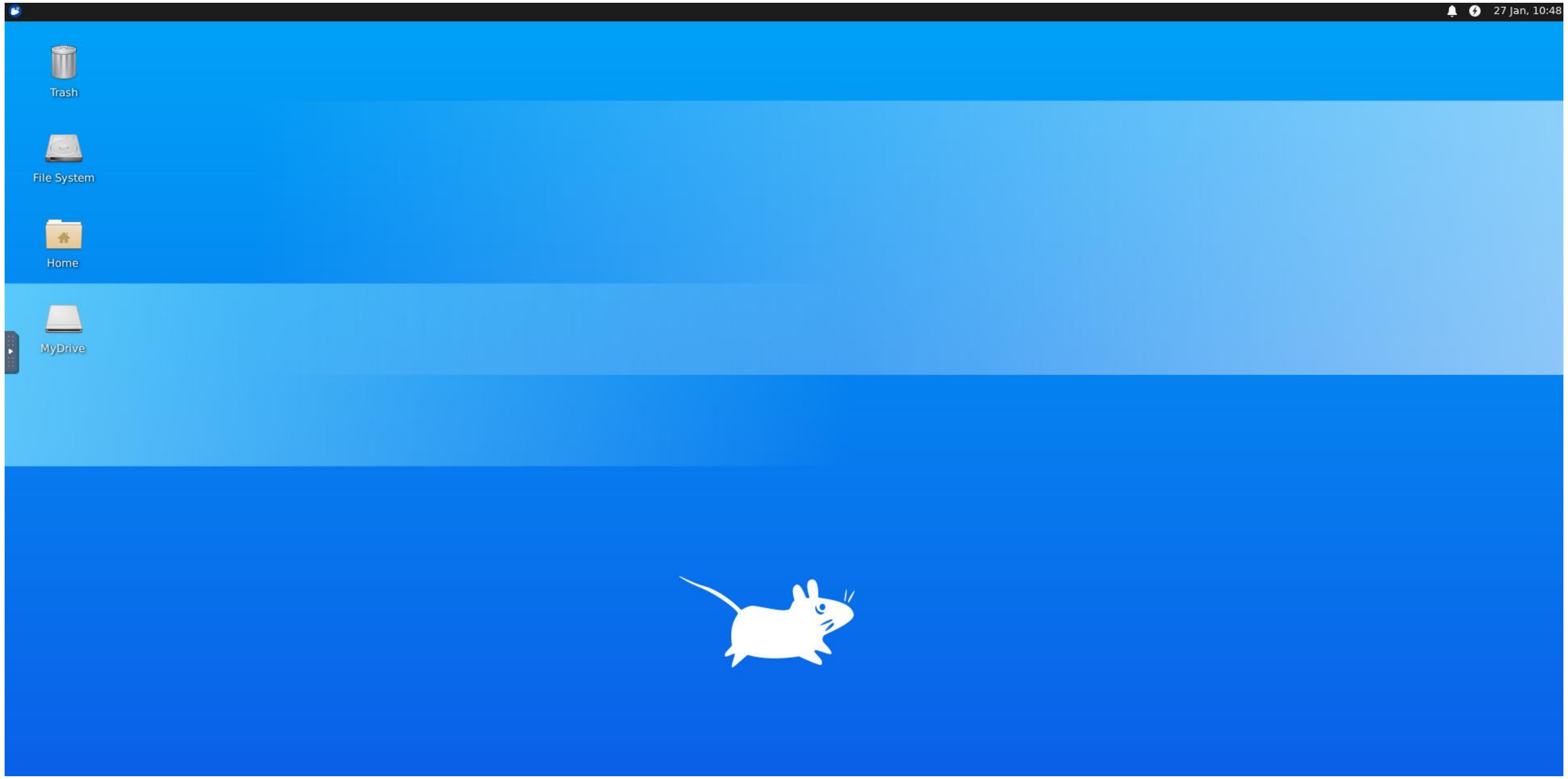
Politecnico di Torino

master school

ABB

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 install 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 recap - 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 <u>NOT</u> 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

```
# 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)
```

1

2

Input file:

```
Lorem ipsum sit\n
amen dolor…
```

```
["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

Launch a Notebook with PySpark using

```
$> pyspark
```

Data Science and AI for industrial systems

Hands-on session 1.2
Towards SparkSQL

Prof. Daniele Apiletti, Simone Monaco

PySpark

17/02/2023

Politecnico di Torino
master school
ABB

Bike sharing dataset in Barcellona

register.csv

station	timestamp	used_slots	free_slots
1	15/05/2008 12:01	0	18
1	15/05/2008 12:02	0	18
1	15/05/2008 12:04	0	18
1	15/05/2008 12:06	0	18
1	15/05/2008 12:08	0	18
1	15/05/2008 12:10	0	18
1	15/05/2008 12:12	0	18
1	15/05/2008 12:14	0	18
1	15/05/2008 12:16	0	18
1	15/05/2008 12:18	0	18
1	15/05/2008 12:20	2	16
1	15/05/2008 12:22	3	15

3 used slots and **15** free slots at station 18 on **May 15, 2008** at **12:22:00**

stations.csv

id	longitude	latitude	name
1	2.180019	41.397978	Gran Via Corts Catalanes
2	2.176414	41.394381	Plaza TetuÃ¡n
3	2.181164	41.39375	Ali Bei
4	2.1814	41.393364	Ribes
5	2.180214	41.391072	Pg LluÃs Companys
6	2.180508	41.391272	Pg LluÃs Companys
7	2.183183	41.388867	Pg LluÃs Companys
8	2.183453	41.389044	Passeig lluÃs companys
9	2.185294	41.385006	MarquÃ¨s de l\'Argentera
10	2.185206	41.384875	Avinguda del Marques Argentera

WARNING: wrong values

Some of the lines of **register.csv** contain wrong data. Specifically, they are characterized by **used slots = 0** and **free slots = 0**. Those lines must be filtered before performing the analysis.

register.csv

station	timestamp	used_slots	free_slots
	...		
15	15/05/2008 12:20	0	0

Task 1

- Write a single Spark application that identifies the most “critical” timeslot for each station.
- Each combination “day of the week – hour” is a timeslot and is associated with all the readings associated with that combination, independently of the date

“Wednesday - 15” : all the readings made on every Wednesday from 15:00:00 to 15:59:59

Task 1

- A station S_i is in the critical state in a specific timestamp if the number of free slots is equal to 0 (i.e., the station is full).
- Compute the $criticality$ as

$$\frac{\{num\ readings\ with\ free\ slot = 0\}(S_i, T_j)}{\{total\ num\ readings\}(S_i, T_j)}$$

Task 1

- Computes the criticality value for each pair (S_i, T_j).
- Selects only the pairs with a criticality value greater than or equal to a **minimum criticality threshold**.
- Selects the **most critical timeslot** for each station. If there are two or more timeslots characterized by the highest criticality value for the same station, select only one of those timeslots (the one associated with the earliest hour).
- Stores in one single (KML) file the information about the most critical timeslot for each station.
  - one marker of type **Placemark** for each pair (S_i, most critical timeslot for S_i) characterized by the following features:

$$\{StationId, Day - hour, Criticality\ value, longitude, latitude\}$$

Task 1

- The output (KML) file must have the following format (one KML Placemark per line):

```
<Placemark><name>44</name>
    <ExtendedData>
        <Data name="DayWeek"><value>Mon</value></Data>
        <Data name="Hour"><value>3</value></Data>
        <Data name="Criticality"><value>0.5440729483282675</value></Data>
    </ExtendedData><Point><coordinates>2.189700,41.379047</coordinates></Point>
</Placemark>
<Placemark><name>9</name>
    <ExtendedData>
        <Data name="DayWeek"><value>Sat</value></Data>
        <Data name="Hour"><value>10</value></Data>
        <Data name="Criticality"><value>0.5215827338129496</value></Data>
    </ExtendedData><Point><coordinates>2.185294,41.385006</coordinates></Point>
</Placemark>
```

Task 1

- The output (KML) file must have the following format (one KML Placemark per line):

```
<Placemark><name>44</name>
    <ExtendedData>
        <Data name="DayWeek"><value>Mon</value></Data>
        <Data name="Hour"><value>3</value></Data>
        <Data name="Criticality"><value>0.5440729483282675</value></Data>
    </ExtendedData><Point><coordinates>2.189700,41.379047</coordinates></Point>
</Placemark>
<Placemark><name>9</name>
    <ExtendedData>
        <Data name="DayWeek"><value>Sat</value></Data>
        <Data name="Hour"><value>10</value></Data>
        <Data name="Criticality"><value>0.5215827338129496</value></Data>
    </ExtendedData><Point><coordinates>2.185294,41.385006</coordinates></Point>
</Placemark>
```

Bonus Task: repeat with Spark SQL