Lab 8

In this lab, we analyze historical data about the stations of the bike sharing system of Barcelona. The data are the same you already analyzed during the previous practice, and also the goal of your task is similar: computing the "criticality" for the pairs (station, timeslot) and select the most critical ones. However, in this practice you are requested to exploit the Spark SQL APIs.

The analysis is based on 2 files available in the HFDS shared folder of the BigData@Polito cluster:

1. /data/students/bigdata-01QYD/Lab7/**register.csv**
2. /data/students/bigdata-01QYD/Lab7/**stations.csv**

1) **register.csv** contains the historical information about the number of used and free slots for ~3000 stations from May 2008 to September 2008. Each line of register.csv corresponds to one reading about the situation of one station at a specific timestamp. Each line has the following format:

   - *station***\t***timestamp***\t***used_slots***\t***free_slots*

   For example, the line

   23 2008-05-15 19:01:00 5   13

   means that there were **5** used slots and **13** free slots at station **23** on **May 15, 2008** at **19:01:00**.

   The first line of register.csv contains the header of the file.

   Pay attention that some of the lines of register.csv contain wrong data due to temporary problems of the monitoring system. Specifically, some lines are characterized by used_slots = 0 and free_slots = 0. Those lines must be filtered before performing the analysis.

2) **stations.csv** contains the description of the stations. Each line of registers.csv has the following format:

   - *id***\t***longitude***\t***latitude***\t***name*

   For example, the line

   1   2.180019   41.397978   Gran Via Corts Catalanes

   contains the information about station **1**. The coordinates of station 1 are 2.180019,41.397978 and its name is **Gran Via Corts Catalanes**.

Ex. 1

Write a single Spark application that selects the pairs (station, timeslot) that are characterized by a high “criticality” value. The first part of this practice is similar to the one that you already solved during the previous practice. However, in this case you are requested to **solve the problem using the Spark SQL APIs.**
You can use one of the following options:

- **(i)** **Typed Datasets,** whenever possible, and the associated type-safe transformations, i.e., map, filter, etc.
- **(ii)** **DataFrames (i.e., Datasets<Row>)** and the associated non type-safe transformations, i.e., select, filter, etc.
- **(iii)** **SQL queries** in the Spark application**,** i.e., SparkSession.sql(“SELECT …”).

Select the option that you prefer.

In this application, each pair “day of the week – hour” is a timeslot and is associated with all the readings associated with that pair, independently of the date. For instance, the timeslot “Wednesday - 15” corresponds to all the readings made on Wednesday from 15:00:00 to 15:59:59.

A station Si is in the critical state if the number of free slots is equal to 0 (i.e., the station if full).
The “criticality” of a station Si in the timeslot Tj is defined as

$$\frac{number\ of\ readings\ with\ num.of\ free\ slot\ equal\ to\ 0\ for\ the\ pair\ (Si,Tj)}{total\ number\ of\ readings\ for\ the\ pair\ (Si,Tj)}$$

Write an application, based on the Spark SQL APIs, that:

- Removes the lines with used_slots = 0 and free_slots = 0.
- Computes the **criticality value** for each pair (Si,Tj).
- Selects only the pairs having a criticality value greater than a minimum criticality threshold. The **minimum criticality threshold** is an argument of the application.
- **Join** the content of the previous selected “records” with the content of stations.csv to retrieve **the coordinates of the stations**.
- Store in the output folder the selected records, by using **csv files (with header)**. Store only the following attributes:
    - station
    - day of week
    - hour
    - criticality
    - station longitude
    - station latitude
- Store the results **by decreasing criticality**. If there are two or more records characterized by the same criticality value, consider the station (in ascending order). If also the station is the same, consider the day of the week (ascending) and finally the hour (ascending).

Note that:

- The provided template includes the class DateTool characterized by the following static methods:
  - **`String DayOfTheWeek(String timestamp)`**
    The provided method returns the day of the week of the String containing a timestamp provides as parameter.
    For instance, the instruction DateTool.DayOfTheWeek("2017-05-12") returns the string "Fri".
  - **`String DayOfTheWeek(java.sql.Timestamp timestamp)`**
    This method returns the same information returned by the other method but the parameter of this version of the method is a java.sql.Timestamp object.
  - **`int hour(java.sql.Timestamp timestamp)`**
    This method returns the hour information associated with the input timestamp.

- The SQL-like language available in Spark SQL is characterized by a predefined function called **hour(timestamp)** that can be used in the SQL queries, or in the **selectExpr** transformation, to select the “hour part” of a given timestamp.
  The **DateTool.hour(Timestamp timestamp)** is needed only in the map transformations associated with typed Datasets.

- To specify that the separator of the input CSV files is “tab”, set the delimiter option to **\\t**, i.e., invoke **.option("delimiter", "\\t")** during the reading of the input data.
- To specify the timestamp format when reading the input CSV file, set the timestampFormat option to **"yyyy-MM-dd HH:mm:ss"**, i.e., invoke **.option("timestampFormat","yyyy-MM-dd HH:mm:ss")**
- Example

  -
    ```
    Dataset<Row> inputDF = ss.read().format("csv")
                    .option("delimiter", "\\t")
                    .option("timestampFormat","yyyy-MM-dd HH:mm:ss")
                    .option("header", true)
                    .option("inferSchema", true)
                    .load(inputPath);
    ```

How to access logs files
If you are connecting from outside Polito and you submit your application on the cluster by using spark-submit you can proceed as follows to retrieve the log files from the command line:
1. Open a Terminal on the gateway jupyter.polito.it
2. Execute the following command in the Terminal:
   yarn logs -applicationId application_1521819176307_2195

The last parameter is the application/job ID. You can retrieve the job ID of your application following one of these procedures:

- Submit the job (with the flag --deploy-mode **client** on the spark-submit command) and adding the following command at the beginning of your application

```
System.out.println("ApplicatioId:"+JavaSparkContext.toSparkContext(sc).applicationId());
```

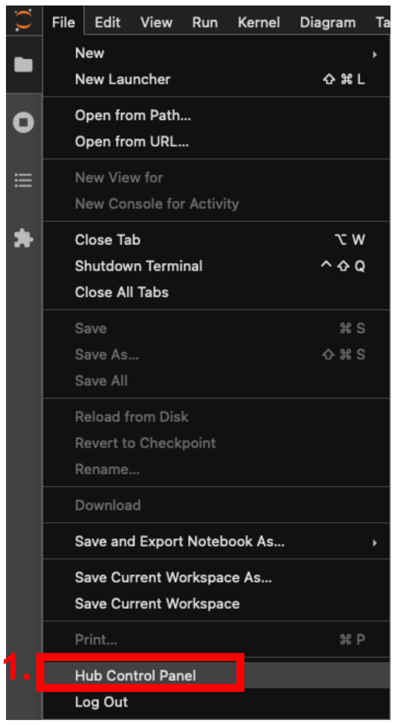
- With the following command on the terminal, substituting **sXXXXX** with your username

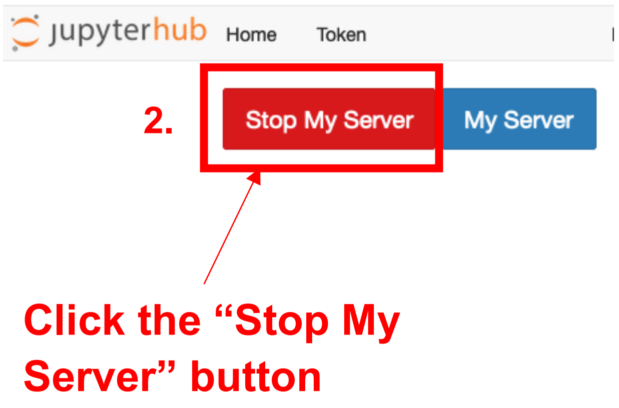
```
yarn application -list -appStates ALL|grep 'sXXXXXX'
```

⚠️⚠️⚠️ Shut down JupyterHub container ⚠️⚠️⚠️

As soon as you complete all the tasks and activities on JupyterHub environment, please remember to shut down the container to let all your colleagues in all the sessions connect on JupyterHub and do all the lab activities.

1. Go into File -> Hub Control Panel menu
2. A new browser tab opens with the "Stop My Server" button. Click on it and wait till it disappears.


File
Edit
View
Run
Kernel
Diagram
New
New Launcher ⇧ ⌘ L
Open from Path...
Open from URL...
New View for
New Console for Activity
Close Tab ⌥ W
Shutdown Terminal ^ ⇧ Q
Close All Tabs
Save ⌘ S
Save As... ⇧ ⌘ S
Save All
Reload from Disk
Revert to Checkpoint
Rename...
Download
Save and Export Notebook As...
Save Current Workspace As...
Save Current Workspace
Print... ⌘ P
1.
Hub Control Panel
Log Out


jupyterhub
Home
Token
2.
Stop My Server
My Server
Click the "Stop My Server" button