Exploring SparkSQL and Spark DataFrames

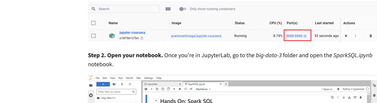
By the end of this activity, you will be able to:

1. Access Postgres database tables with SparkSQL
2. Filter rows and columns of a Spark DataFrame
3. Group and perform aggregate functions on columns in a Spark DataFrame
4. Join two SparkDataFrames on a single column

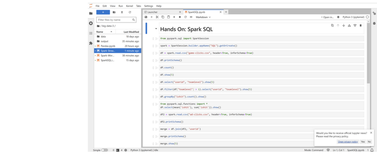
For this activity, you should have completed the creation of the JupyterLab container. If not follow, Steps 1-3 on the previous activity *Hand On: Exploring Pandas DataFrames*, and then come back to Step 2 of this activity.

Step 1. Start the container. Open Docker Desktop and start your *jupyter-coursera* container.

When Jupyter starts running, click on the port to access JupyterLab in your browser:



Step 2. Open your notebook. Once you're in JupyterLab, go to the *big-data-3* folder and open the *SparkSQL.ipynb* notebook.



Step 3. Connect to file. The first line imports the SparkSession module, which is the entry point to start working with data and DataFrames.

```
[1]: from pyspark.sql import SparkSession
```

The second line creates a new SparkSession:

```
[2]: spark = SparkSession.builder.appName("SQL").getOrCreate()
```

The third line creates a new Spark DataFrame in the variable *df* for the csv file *gameclicks* (this database was also available in the PostgresSQL activity):

```
[3]: df = spark.read.csv("data/game-clicks.csv", header=True, inferSchema=True)
```

Step 4. View Spark DataFrame schema and count rows. We can call the *printSchema()* method to view the schema of the DataFrame:

```
[4]: df.printSchema()
```

```
root
 |-- timestamp: timestamp (nullable = true)
 |-- clickId: integer (nullable = true)
 |-- userId: integer (nullable = true)
 |-- userSessionId: integer (nullable = true)
 |-- isHit: integer (nullable = true)
 |-- teamId: integer (nullable = true)
 |-- teamLevel: integer (nullable = true)
```

The description lists the name and data type of each column.

We can also call the *count()* method to count the number of rows in the DataFrame:

```
[5]: df.count()
```

```
[5]: 755806
```

Step 5. View contents of DataFrame. We can call the *show()* method to view the contents of the DataFrame. The argument specifies how many rows to display:

```
[6]: df.show(5)
```

```
+-------------------+-------+------+-------------+-----+--------+---------+
|          timestamp|clickId|userId|userSessionId|isHit|teamId|teamLevel|
+-------------------+-------+------+-------------+-----+--------+---------+
|2016-05-26 15:06:55|    105|  1038|         5916|    0|    25|        1|
|2016-05-26 15:07:09|    154|  1099|         5898|    0|    44|        1|
|2016-05-26 15:07:14|    229|   899|         5757|    0|    71|        1|
|2016-05-26 15:07:14|    322|  2197|         5854|    0|    99|        1|
|2016-05-26 15:07:20|    221|  1362|         5739|    0|    13|        1|
+-------------------+-------+------+-------------+-----+--------+---------+
only showing top 5 rows
```

Step 6. Filter columns in DataFrame. We can filter for one or more columns by calling the *select()* method:

```
[7]: df.select("userid", "teamlevel").show(5)
```

```
+------+---------+
|userid|teamlevel|
+------+---------+
|  1038|        1|
|  1099|        1|
|   899|        1|
|  2197|        1|
|  1362|        1|
+------+---------+
only showing top 5 rows
```

Step 7. Filter rows based on criteria. We can also filter for rows that match a specific criteria using *filter()*:

```
[8]: df.filter(df["teamlevel"] > 1).select("userid", "teamlevel").show(5)
```

```
+------+---------+
|userid|teamlevel|
+------+---------+
|  1513|        2|
|   868|        2|
|  1453|        2|
|  1282|        2|
|  1473|        2|
+------+---------+
only showing top 5 rows
```

The arguments to *filter()* are a Column, in this case specified as *df["teamlevel"]*, and the condition, which is greater than 1. The remainder of the commander selects only the *userid* and *teamlevel* columns and shows the first five rows.

Step 8. Group by a column and count. The *groupBy()* method groups the values of column(s). The *ishit* column only has values 0 and 1. We can calculate how many times each occurs by grouping the *ishit* column and counting the result:

```
[9]: df.groupBy("ishit").count().show()
```

```
+-----+------+
|ishit| count|
+-----+------+
|    1| 83383|
|    0|672423|
+-----+------+
```

Step 9. Calculate average and sum. Aggregate operations can be performed on columns of DataFrames. First, let's import the Python libraries for the aggregate operations. Next, we can calculate the average and total values by calling the *mean()* and *sum()* methods, respectively:

```
[10]: from pyspark.sql.functions import *
      df.select(mean('ishit'), sum('ishit')).show()
```

```
+-----------------+----------+
|        avg(ishit)|sum(ishit)|
+-----------------+----------+
|0.1103232840173272|     83383|
+-----------------+----------+
```

Step 10. Join two DataFrames. We can merge or join two DataFrames on a single column. First, let's create a DataFrame for the *adclicks* table storing the result in a new variable *df2*:

```
[11]: df2 = spark.read.csv("data/ad-clicks.csv", header=True, inferSchema=True)
```

Let's view the columns in *df2* by calling *printSchema()*:

```
[12]: df2.printSchema()
```

```
root
 |-- timestamp: timestamp (nullable = true)
 |-- txId: integer (nullable = true)
 |-- userSessionId: integer (nullable = true)
 |-- teamId: integer (nullable = true)
 |-- userId: integer (nullable = true)
 |-- adId: integer (nullable = true)
 |-- adCategory: string (nullable = true)
```

We can see that the *adclicks df2* DataFrame also has a column called *userid*. Next, we will combine the *gameclicks* and *adclicks* DataFrames by calling the *join()* method and saving the resulting DataFrame in a variable called *merge*:

```
[13]: merge = df.join(df2, 'userid')
```

We are calling the *join()* method on the *gameclicks* DataFrame; the first argument is the DataFrame to join with, i.e., the *adclicks* DataFrame, and the second argument is the column name in both DataFrames to join on.

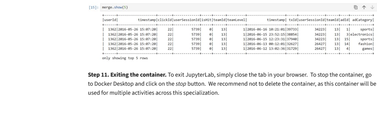
Let's view the schema of *merge*:

```
[14]: merge.printSchema()
```

```
root
 |-- userId: integer (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- clickId: integer (nullable = true)
 |-- userSessionId: integer (nullable = true)
 |-- isHit: integer (nullable = true)
 |-- teamId: integer (nullable = true)
 |-- teamLevel: integer (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- txId: integer (nullable = true)
 |-- userSessionId: integer (nullable = true)
 |-- teamId: integer (nullable = true)
 |-- adId: integer (nullable = true)
 |-- adCategory: string (nullable = true)
```

We can see that the merged DataFrame has all the columns of both *gameclicks* and *adclicks*.

Finally, let's look at the contents of *merge*:



Step 11. Exiting the container. To exit JupyterLab, simply close the tab in your browser. To stop the container, go to Docker Desktop and click on the *stop* button. We recommend not to delete the container, as this container will be used for multiple activities across this specialization.

Go to next item ✓ Completed