



## **Practical F02**

### **Data Analysis using Spark SQL**

Objectives of this practical

1. Import JSON dataset
2. Perform Data Analysis using Spark SQL
3. Perform Data Analysis using the Window function

## Section 1 About the Practical

In this practical, we are going to explore 'CPU Utilization' data using SPARK SQL.

The dataset we use, the 'CPU Utilization' data is a kind of **Time Series data**. Time Series data is data that has a set of measures and a timestamp associated with the time. We are going to explore the time series data using **Window functions**.

Before you begin, please download the dataset from the module web site. It is a folder consists of four JSON files.

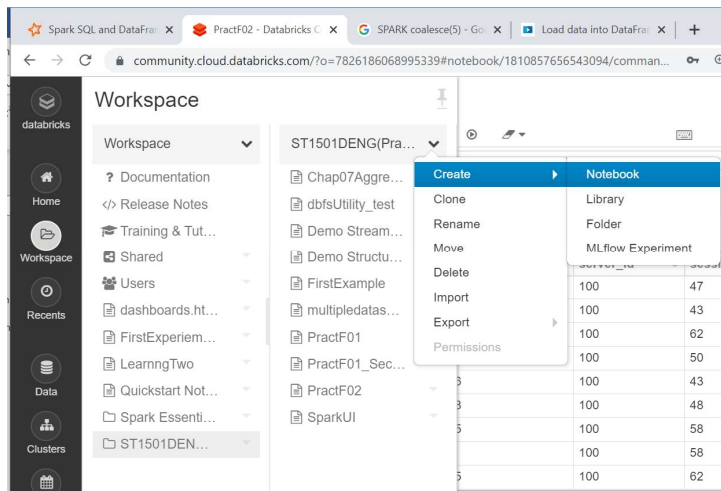
Fig: The CPU Utilization Data

cpu_utilization	event_datetime	free_memory	server_id	session_count
0.57	03/05/2019 08:06:14	0.51	100	47
0.47	03/05/2019 08:11:14	0.62	100	43
0.56	03/05/2019 08:16:14	0.57	100	62
0.57	03/05/2019 08:21:14	0.56	100	50
0.35	03/05/2019 08:26:14	0.46	100	43
0.41	03/05/2019 08:31:14	0.58	100	48
0.57	03/05/2019 08:36:14	0.35	100	58
0.41	03/05/2019 08:41:14	0.4	100	58
0.53	03/05/2019 08:46:14	0.35	100	62

## Section 2 Data Analysis using Spark SQL

### Task: Upload and Read the 'CPU Utilization' data (four json files)

- a) Create a new SQL Notebook, named it as 'PractF02' within the 'ST1501DENG' folder.



- b) In the 'PractF02' Notebook, enter the following command to create a new folder 'utilization'. The new folder is to store our 'CPU utilization' data from four JSON files.

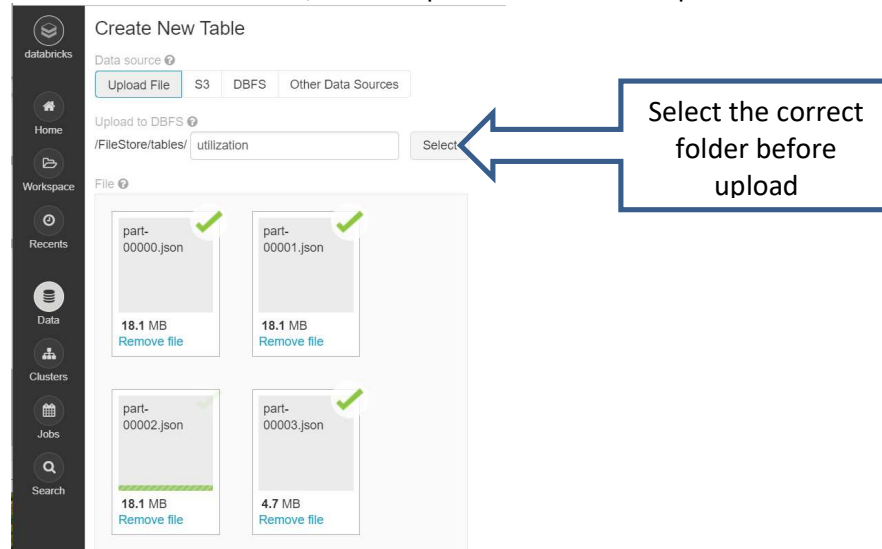
Cmd 1

```
1 %python
2 dbutils.fs.mkdirs("dbfs:/FileStore/tables/utilization")
```

After you have successfully executed the command, use '#' preceding line 2 to comment out the line.

- c) Switch back to the Databricks UI. You are going to import all the four JSON files to the new folder.

Select the correct folder, then drop files to the File dropzone.



- d) Switch back to the PractF02 Notebook and enter the following command to check the files are successfully uploaded.

```
1 %python
2 # Check all the JSON Files are uploaded
3 dbutils.fs.ls("/FileStore/tables/utilization")
```

```
Out[6]: [FileInfo(path='dbfs:/FileStore/tables/utilization/part_00000-6a99e.json', name='part_00000-6a99e.json', size=18059770),
FileInfo(path='dbfs:/FileStore/tables/utilization/part_00001-f7f45.json', name='part_00001-f7f45.json', size=18067375),
FileInfo(path='dbfs:/FileStore/tables/utilization/part_00002-d3a56.json', name='part_00002-d3a56.json', size=18064544),
FileInfo(path='dbfs:/FileStore/tables/utilization/part_00003-97792.json', name='part_00003-97792.json', size=4713435)]
```

Command took 0.10 seconds -- by leong\_fong\_sow@sp.edu.sg at 12/15/2019, 10:46:33 PM on My Cluster

- e) Enter the following command to create the utilization table.

Cmd 4

```
1 -- The Create statement should only run once
2 -- The schema is infer using the JSON key-values pairs
3 CREATE TEMPORARY TABLE utilization
4 USING JSON
5 OPTIONS (path "/FileStore/tables/utilization");
```

You have just executed a powerful command to create a table of 500,000 records from a folder that consists of multiple JSON files.

Check that 'utilization' table is created by using a SELECT statement.

**Task: Exploration Data Analysis using Spark SQL**

- a) From the previous practical, you should notice that Spark SQL supports the standard SQL. Now write the SQL statement to answer the following questions.

Q1. How many CPU Utilization records do we have?

Q2. List all the server\_id, without duplicates.

Q3. How many server does we have?

Q4. Show all the CPU utilization records related to server\_id = 120 and session\_count is more than 70.

Q5. What is the number of events that the CPU\_utilization is lower than 0.3 of each server?

Q6. What is the minimum, maximum and standard deviation CPU utilization of each server? The output for Q6 should like the following.

► (1) Spark Jobs

server_id	min(cpu_utilization)	max(cpu_utilization)	stddev_samp(cpu_utilization)
100	0.27	0.67	0.1152264191787964
101	0.6	1	0.11651726263197697
102	0.56	0.96	0.11549678751286807
103	0.56	0.96	0.11617507884178278
104	0.51	0.91	0.11521679513850511
105	0.29	0.69	0.11510721467869486
106	0.22	0.62	0.11531539914568233

### Task: Create a Bar Chart/Histogram of CPU Utilization

- a) In this task, we want to know how often a CPU utilization falls in a certain range. We will present the information using a bar chart.


Our CPU utilization is a decimal number ranges from 0 to 1. We are going to have 10 buckets. Since we have a decimal value, we will multiply it by 100 and then divide by 10. It will create range (1-10), (11-20) and etc.

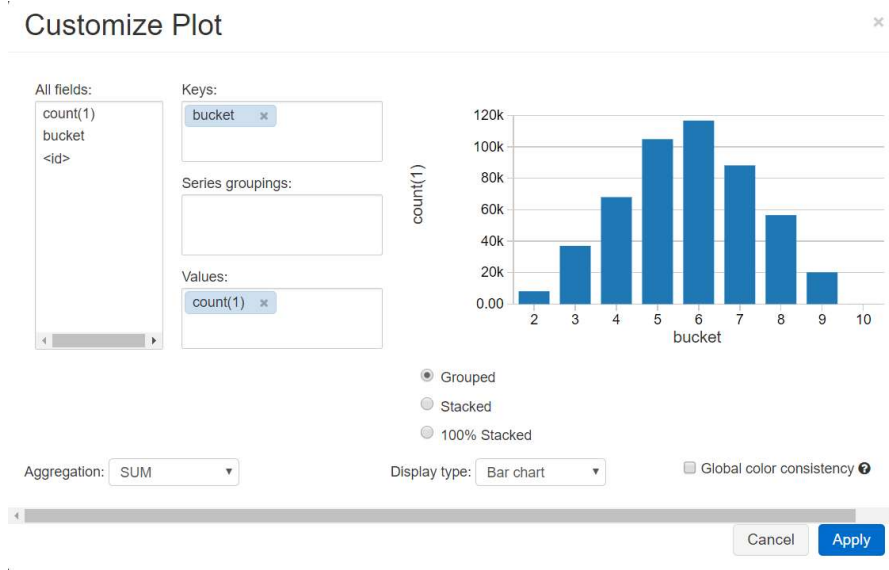
Try out the following SQL statement to understand the **floor** function.

```
SELECT server_id, FLOOR(cpu_utilization*100/10) bucket FROM utilization
```

Enter the following code to create the data required for the chart.



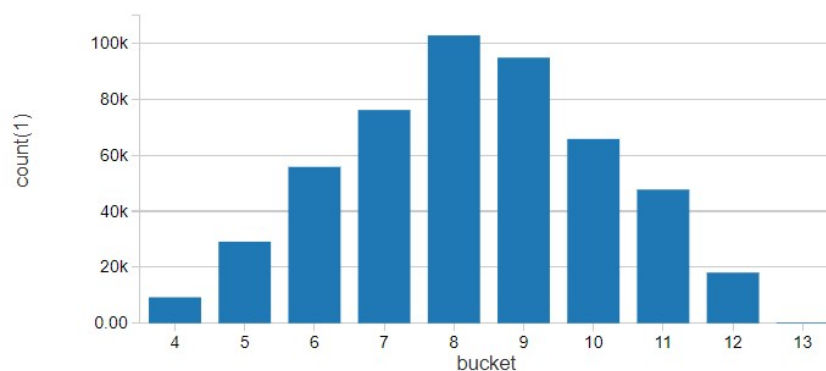
- b) Click on the  and click the 'Plot Option'. Adjust the required parameters (x and y values) to plot the bar chart. Click 'Apply'.



- c) Now, we want to know how the session count falls in a certain range. From the following query, we know the session count falls between 32 to 105.

```
SELECT min(session_count), max(session_count) FROM utilization
```

We again wish to present the information using histogram, with 10 buckets. Write the Spark SQL to create a histogram similar to the following.



## Section 3 Data Analysis using the Window Function

### No. Task

- a) We want to compare the CPU utilization to the average CPU utilization of just that server, not the entire population.

To compute the average CPU utilization of each server, you can easily do it using the Group By clause. However, the Group By clause alone will not produce the required information.

```
1 SELECT server_id, avg(cpu_utilization)
2 FROM utilization group by server_id order by server_id
```

In following table, we have varying timestamps, for example here is one at about 8:06, 8:11, 8:21 and so on all for server ID 112. We have different CPU utilizations at those particular times but the average server utilization is always 0.71538.

event_datetime	server_id	cpu_utilization	avg_server_util
03/05/2019 08:06:34	112	0.71	0.7153870000000067
03/05/2019 08:11:34	112	0.78	0.7153870000000067
03/05/2019 08:16:34	112	0.87	0.7153870000000067
03/05/2019 08:21:34	112	0.82	0.7153870000000067
03/05/2019 08:26:34	112	0.62	0.7153870000000067
03/05/2019 08:31:34	112	0.9	0.7153870000000067

Showing the first 1000 rows.

The above table can be done using the **Window function**. We can use the window function to carry out some unique aggregation by aggregating some aggregation on a specific 'window'.

- A group-by takes data, and every row can go only into one grouping.
- A window function calculates *a return value for every input row* for a table based on a group of rows.

Enter the following command. You should get a table similar to the above.

```
SELECT event_datetime, server_id, cpu_utilization,
       avg(cpu_utilization) OVER (PARTITION BY server_id) avg_server_util
FROM utilization
```



- b) To compare the CPU utilization to the average CPU utilization, we can now add another column to do the arithmetic, subtract the two numbers. Name the new column as 'delta\_server\_util'.

Modify the preceding SELECT statement to calculate the required. The table output should be similar to the following.

event_datetime	server_id	cpu_utilization	avg_server_util	delta_server_util
03/05/2019 08:06:34	112	0.71	0.7153870000000067	-0.005387000000006692
03/05/2019 08:11:34	112	0.78	0.7153870000000067	0.06461299999999337
03/05/2019 08:16:34	112	0.87	0.7153870000000067	0.15461299999999334
03/05/2019 08:21:34	112	0.82	0.7153870000000067	0.1046129999999933
03/05/2019 08:26:34	112	0.62	0.7153870000000067	-0.09538700000000666
03/05/2019 08:31:34	112	0.9	0.7153870000000067	0.18461299999999337
03/05/2019 08:36:34	112	0.89	0.7153870000000067	0.17461299999999336
03/05/2019 08:41:34	112	0.81	0.7153870000000067	0.0946129999999934

Showing the first 1000 rows.

- c) We can also use the Window function to compare a particular value in a row to a value of some aggregate function applied to a *sub-set of rows*. It means we can use Window function look around essentially the neighborhood of a row. For example, look at the last three values and average them or look at the last value or current value, next value and average them.

Enter the following command to understand how it works.

```
SELECT event_datetime, server_id, cpu_utilization,
       avg(cpu_utilization) OVER (PARTITION BY server_id ORDER BY event_datetime
                                ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) avg_server_util
FROM utilization
```

The expected output:

Cmd: 28

```
1 SELECT event_datetime, server_id, cpu_utilization,
2       avg(cpu_utilization) OVER (PARTITION BY server_id ORDER BY event_datetime
3                                ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) avg_server_util
4 FROM utilization
```

1 Spark Jobs

event_datetime	server_id	cpu_utilization	avg_server_util
03/05/2019 08:06:34	112	0.71	0.745
03/05/2019 08:11:34	112	0.78	0.7866666666666666
03/05/2019 08:16:34	112	0.87	0.8233333333333333
03/05/2019 08:21:34	112	0.82	0.77
03/05/2019 08:26:34	112	0.62	0.7799999999999999
03/05/2019 08:31:34	112	0.9	0.8033333333333333

Showing the first 1000 rows.

Calculate:  
 $(0.71+0.78+0.87)/3$

-- End of Practical --