“What are the different aggregate functions in PySpark and can you provide some examples of their usage?

Authored by

stats writer

June 24, 2024

Aggregate functions in PySpark are built-in functions that allow for data manipulation and summarization on large datasets. These functions operate on multiple rows of data, grouping them together and producing a single result for each group. Some examples of aggregate functions in PySpark include sum, count, min, max, and avg.

The sum function, for instance, adds up all the values in a column and returns the total. This can be useful for calculating the total revenue or sales for a company. The count function, on the other hand, counts the number of non-null values in a column, providing insights into the size of a dataset.

Other aggregate functions, such as min and max, return the minimum and maximum values in a column, respectively. These can be helpful in finding the smallest or largest values in a dataset. The avg function calculates the average of a column, useful for determining the average salary or cost of a product.

Overall, aggregate functions in PySpark are essential tools for data analysis and can provide valuable insights into a dataset. They allow for efficient and effective data summarization, making it easier to draw meaningful conclusions and make informed decisions.

Aggregate functions in PySpark are essential for summarizing data across distributed datasets. They allow computations like sum, average, count, maximum, and minimum to be performed efficiently in parallel across multiple nodes in a cluster.

These functions are primarily used with large datasets to get single values or smaller result sets, providing insights into the dataset's characteristics. Aggregate functions operate on a group of rows and calculate a single return value for every group. All these aggregate functions accept input as, Column type or column name as a string and several other arguments based on the function.

PySpark Aggregate Functions

PySpark SQL Aggregate functions are grouped as "agg_funcs" in Pyspark. Below is a list of functions defined under this group. Click on each link to learn with example.

PySpark Aggregate Functions Examples

First, let's create a DataFrame to work with PySpark aggregate functions. All examples provided here are also available at PySpark Examples GitHub project.

```
simpleData =
schema =
```

```
df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)
```

Yields below output.

```
# Output
+-------------+----------+------+
|employee_name|department|salary|
+-------------+----------+------+
| James| Sales| 3000|
| Michael| Sales| 4600|
| Robert| Sales| 4100|
| Maria| Finance| 3000|
| James| Sales| 3000|
| Scott| Finance| 3300|
| Jen| Finance| 3900|
| Jeff| Marketing| 3000|
| Kumar| Marketing| 2000|
| Saif| Sales| 4100|
+-------------+----------+------+
```

Now let's see how to aggregate data in PySpark.

approx_count_distinct Aggregate Function

In PySpark `approx_count_distinct()` function returns the count of distinct items in a group.

```
# approx_count_distinct()
print("approx_count_distinct: " +
str(df.select(approx_count_distinct("salary")).collect()))
```

Prints approx_count_distinct: 6

avg (average) Aggregate Function

`avg()` function returns the average of values in the input column.

```
# avg
```

```
print("avg: " + str(df.select(avg("salary")).collect()))
```

Prints avg: 3400.0

collect_list Aggregate Function

`collect_list()` function returns all values from an input column with duplicates.

```
# collect_list
df.select(collect_list("salary")).show(truncate=False)
```

```
+---------------------------------------------------------+
|collect_list(salary) |
+---------------------------------------------------------+
||
+---------------------------------------------------------+
```

collect_set Aggregate Function

`collect_set()` function returns all values from an input column with duplicate values eliminated.

```
# collect_set
df.select(collect_set("salary")).show(truncate=False)
```

```
+---------------------------------+
|collect_set(salary) |
+---------------------------------+
||
+---------------------------------+
```

countDistinct Aggregate Function

`countDistinct()` function returns the number of distinct elements in a columns

```
# countDistinct
df2 = df.select(countDistinct("department", "salary"))
df2.show(truncate=False)
```

```
print("Distinct Count of Department & Salary: "+str(df2.collect()))
```

count function

`count()` function returns number of elements in a column.

```
print("count: "+str(df.select(count("salary")).collect()))
```

Prints county: 10

grouping function

`grouping()` Indicates whether a given input column is aggregated or not. returns 1 for aggregated or 0 for not aggregated in the result. If you try grouping directly on the salary column you will get below error.

```
Exception in thread "main" org.apache.spark.sql.AnalysisException:
// grouping() can only be used with GroupingSets/Cube/Rollup
```

first function

`first()` function returns the first element in a column when ignoreNulls is set to true, it returns the first non-null element.

```
# first
df.select(first("salary")).show(truncate=False)
```

```
+--------------------+
|first(salary, false)|
+--------------------+
|3000 |
+--------------------+
```

last function

`last()` function returns the last element in a column. when ignoreNulls is set to true, it returns the

last non-null element.

```
# last
df.select(last("salary")).show(truncate=False)
```

```
+------------------+
|last(salary, false)|
+------------------+
|4100 |
+------------------+
```

kurtosis function

`kurtosis()` function returns the kurtosis of the values in a group.

```
df.select(kurtosis("salary")).show(truncate=False)
```

```
+------------------+
|kurtosis(salary) |
+------------------+
|-0.6467803030303032|
+------------------+
```

max function

`max()` function returns the maximum value in a column.

```
df.select(max("salary")).show(truncate=False)
```

```
+-----------+
|max(salary)|
+-----------+
|4600 |
+-----------+
```

min function

`min()` function

```
df.select(min("salary")).show(truncate=False)
```

```
+-----------+
|min(salary)|
+-----------+
|2000 |
+-----------+
```

mean function

`mean()` function returns the average of the values in a column. Alias for Avg

```
df.select(mean("salary")).show(truncate=False)
```

```
+-----------+
|avg(salary)|
+-----------+
|3400.0 |
+-----------+
```

skewness function

`skewness()` function returns the skewness of the values in a group.

```
df.select(skewness("salary")).show(truncate=False)
```

```
+--------------------+
|skewness(salary) |
+--------------------+
|-0.12041791181069571|
+--------------------+
```

stddev(), stddev_samp() and stddev_pop()

`stddev()` alias for `stddev_samp`.

`stddev_samp()` function returns the sample standard deviation of values in a column.

`stddev_pop()` function returns the population standard deviation of the values in a column.

```
df.select(stddev("salary"), stddev_samp("salary"),
stddev_pop("salary")).show(truncate=False)
```

```
+------------------+------------------+-----------------+
|stddev_samp(salary)|stddev_samp(salary)|stddev_pop(salary)|
+------------------+------------------+-----------------+
|765.9416862050705 |765.9416862050705 |726.636084983398 |
+------------------+------------------+-----------------+
```

sum function

`sum()` function Returns the sum of all values in a column.

```
df.select(sum("salary")).show(truncate=False)
```

```
+-----------+
|sum(salary)|
+-----------+
|34000 |
+-----------+
```

sumDistinct function

`sumDistinct()` function returns the sum of all distinct values in a column.

```
df.select(sumDistinct("salary")).show(truncate=False)
```

```
+--------------------+
|sum(DISTINCT salary)|
+--------------------+
```

|20900 |
+--------------------+

variance(), var_samp(), var_pop()

`variance()` alias for `var_samp`

`var_samp()` function returns the unbiased variance of the values in a column.

`var_pop()` function returns the population variance of the values in a column.

```
df.select(variance("salary"),var_samp("salary"),var_pop("salary"))
.show(truncate=False)
```

+-----------------+-----------------+--------------+
|var_samp(salary) |var_samp(salary) |var_pop(salary)|
+-----------------+-----------------+--------------+
|586666.6666666666|586666.6666666666|528000.0 |
+-----------------+-----------------+--------------+

Source code of PySpark Aggregate examples

```
# Imports
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import approx_count_distinct,collect_list
from pyspark.sql.functions import collect_set,sum,avg,max,countDistinct,count
from pyspark.sql.functions import first, last, kurtosis, min, mean, skewness
from pyspark.sql.functions import stddev, stddev_samp, stddev_pop, sumDistinct
from pyspark.sql.functions import variance,var_samp, var_pop
```

spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()

simpleData =
schema =

df = spark.createDataFrame(data=simpleData, schema = schema)
df.printSchema()
df.show(truncate=False)

```
print("approx_count_distinct: " +
str(df.select(approx_count_distinct("salary")).collect()))

print("avg: " + str(df.select(avg("salary")).collect()))

df.select(collect_list("salary")).show(truncate=False)

df.select(collect_set("salary")).show(truncate=False)

df2 = df.select(countDistinct("department", "salary"))
df2.show(truncate=False)
print("Distinct Count of Department & Salary: "+str(df2.collect()))

print("count: "+str(df.select(count("salary")).collect()))
df.select(first("salary")).show(truncate=False)
df.select(last("salary")).show(truncate=False)
df.select(kurtosis("salary")).show(truncate=False)
df.select(max("salary")).show(truncate=False)
df.select(min("salary")).show(truncate=False)
df.select(mean("salary")).show(truncate=False)
df.select(skewness("salary")).show(truncate=False)
df.select(stddev("salary"), stddev_samp("salary"),
stddev_pop("salary")).show(truncate=False)
df.select(sum("salary")).show(truncate=False)
df.select(sumDistinct("salary")).show(truncate=False)
df.select(variance("salary"),var_samp("salary"),var_pop("salary"))
.show(truncate=False)
```

Conclusion

In this article, I've consolidated and listed all PySpark Aggregate functions with Python examples and also learned the benefits of using PySpark SQL functions. Additionally, aggregate functions are often used in conjunction with group-by operations to perform calculations on grouped data. Overall, aggregate functions facilitate data analysis tasks, enabling users to derive meaningful insights and make informed decisions based on the summarized information.

Happy Learning !!

Related Articles