Master PySpark: From Zero to Big Data Hero!!

Aggregate function in Dataframe – Part 2

Let's create some sample data to demonstrate each of these PySpark DataFrame operations and give notes explaining the functions. Here's how you can create a PySpark DataFrame and apply these operations.

Sample Data

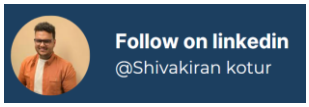
```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType

# Create Spark session
spark =
SparkSession.builder.appName("AggregationExamples").getOrCreate()

# Sample data
data = [
    ("HR", 10000, 500, "John"),
    ("Finance", 20000, 1500, "Doe"),
    ("HR", 15000, 1000, "Alice"),
    ("Finance", 25000, 2000, "Eve"),
    ("HR", 20000, 1500, "Mark")
]

# Define schema
schema = StructType([
    StructField("department", StringType(), True),
    StructField("salary", IntegerType(), True),
    StructField("bonus", IntegerType(), True),
    StructField("employee_name", StringType(), True)
])

# Create DataFrame
df = spark.createDataFrame(data, schema)
df.show()
```



Sample Data Output:

```
+----------+------+-----+-------------+
|department|salary|bonus|employee_name|
+----------+------+-----+-------------+
|        HR| 10000|  500|         John|
|   Finance| 20000| 1500|          Doe|
|        HR| 15000| 1000|        Alice|
|   Finance| 25000| 2000|          Eve|
|        HR| 20000| 1500|         Mark|
+----------+------+-----+-------------+
```

---

1. Grouped Aggregation

Perform aggregation within groups based on a grouping column.

```
df.groupBy("department").agg(
    F.sum("salary"),
    F.avg("salary"),
    F.max("salary"),
    F.min("salary")
).show()
```

▸ (2) Spark Jobs

```
+----------+-----------+-----------+-----------+-----------+
|department|sum(salary)|avg(salary)|max(salary)|min(salary)|
+----------+-----------+-----------+-----------+-----------+
|        HR|      45000|    15000.0|      20000|      10000|
|   Finance|      45000|    22500.0|      25000|      20000|
+----------+-----------+-----------+-----------+-----------+
```

Explanation:

- sum: Adds the values in the group for column1.
- avg: Calculates the average value of column1 in each group.
- max: Finds the maximum value.
- min: Finds the minimum value.

---

2. Multiple Aggregations

Perform multiple aggregations in a single step.

```
df.groupBy("department").agg(
    F.count("salary"),
    F.avg("bonus"),
    F.max("salary")
).show()
```

▸ (2) Spark Jobs

```
+----------+-------------+----------+-----------+
|department|count(salary)|avg(bonus)|max(salary)|
+----------+-------------+----------+-----------+
|        HR|            3|    1000.0|      20000|
|   Finance|            2|    1750.0|      25000|
+----------+-------------+----------+-----------+
```

Explanation:

- count: Counts the number of rows in each group.
- avg: Computes the average of column2.
- max: Finds the maximum value in column1.

---

3. Concatenate Strings

Concatenate strings within a column.

```
df.agg(F.concat_ws(", ", F.collect_list("employee_name")).alias("concatenated_names")).show(truncate=False)
```

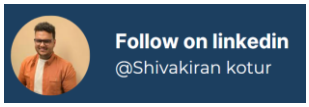
▸ (2) Spark Jobs

```
+--------------------------+
|concatenated_names        |
+--------------------------+
|John, Doe, Alice, Eve, Mark|
+--------------------------+
```

```
df.groupBy("department").agg(
    F.concat_ws(", ", F.collect_list("employee_name")).alias("concatenated_names")
).show(truncate=False)
```

▸ (2) Spark Jobs

```
+----------+------------------+
|department|concatenated_names|
+----------+------------------+
|HR        |John, Alice, Mark |
|Finance   |Doe, Eve          |
+----------+------------------+
```



Follow me on LinkedIn – Shivakiran kotur

Explanation:

- concat_ws: Concatenates string values within the column, separating them by the specified delimiter (,).

---

4. First and Last

Find the first and last values in a column (within each group).

```
df.groupBy("department").agg(F.first("employee_name"), F.last("employee_name")).show()
```

▸ (2) Spark Jobs

```
+----------+--------------------+-------------------+
|department|first(employee_name)|last(employee_name)|
+----------+--------------------+-------------------+
|   Finance|                 Doe|                Eve|
|        HR|                John|               Mark|
+----------+--------------------+-------------------+
```

Explanation:

- first: Retrieves the first value of the name column within each group.
- last: Retrieves the last value of the name column within each group.

---

5. Standard Deviation and Variance

Calculate the standard deviation and variance of values in a column.

```
df.select(F.stddev("salary"), F.variance("salary")).show()
```

▸ (2) Spark Jobs

```
+------------------+----------------+
|stddev_samp(salary)|var_samp(salary)|
+------------------+----------------+
|   5700.87712549569|          3.25E7|
+------------------+----------------+
```

Explanation:

- stddev: Calculates the standard deviation of column.
- variance: Calculates the variance of column.

---

6. Aggregation with Alias

Provide custom column names for the aggregated results.

```
df.groupBy("department").agg(
    F.sum("salary").alias("total_salary"),
    F.avg("salary").alias("average_salary")
).show()
```

▸ (2) Spark Jobs

```
+----------+------------+--------------+
|department|total_salary|average_salary|
+----------+------------+--------------+
|        HR|       45000|       15000.0|
|   Finance|       45000|       22500.0|
+----------+------------+--------------+
```

Explanation:

- .alias(): Used to rename the resulting columns from the aggregation.

---

7. Sum of Distinct Values

Calculate the sum of distinct values in a column.

```
df.select(F.sumDistinct("salary")).show()
```

▸ (3) Spark Jobs

```
+--------------------+
|sum(DISTINCT salary)|
+--------------------+
|               70000|
+--------------------+
```

Explanation:

- sumDistinct: Sums only the distinct values in column. This avoids counting duplicates.

These examples showcase various aggregation operations in PySpark, useful in data summarization and analysis. The grouped aggregation functions like sum(), avg(), and max() are frequently used in big data pipelines to compute metrics for different segments or categories.