Master PySpark: From Zero to Big Data Hero!!

Windows Function in PySpark

1. Introduction to Window Functions

Window functions allow you to perform calculations across a set of rows related to the current row within a specified partition. Unlike groupBy functions, window functions do not reduce the number of rows in the result; instead, they calculate a value for each row based on the specified window.

2. Importing Required Libraries

To use window functions, import the necessary modules from PySpark:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window
```

3. Creating a Window Specification

A window specification defines how the rows will be grouped (partitioned) and ordered within each group.

Example: Basic Window Specification

```
window_spec = Window.partitionBy("category").orderBy("timestamp")
```

Example: Advanced Window Specification with Multiple Partition and Order Columns

```
window_spec = Window.partitionBy("category",
"sub_category").orderBy(F.col("timestamp"), F.col("score"))
```

4. Common Window Functions

Here are several commonly used window functions with explanations and examples:

a. Row Number

- **Function**: row_number()
- **Description**: Assigns a unique integer to each row within the partition. The numbering starts from 1.
- **Example**

```
df = df.withColumn("row_number", F.row_number().over(window_spec))
```

b. Rank

- **Function**: rank()
- **Description**: Assigns the same rank to rows with the same values in the order criteria. The next rank has a gap.
- **Example**:

```
df = df.withColumn("rank", F.rank().over(window_spec))
```

c. Dense Rank

- **Function**: dense_rank()
- **Description**: Similar to rank(), but does not leave gaps in the ranking.
- **Example**:

```
df = df.withColumn("dense_rank",F.dense_rank().over(window_spec))
```

d. Lead and Lag Functions

- **Functions**: lead(), lag()
- **Description**:
  - lead() returns the value of the next row within the window.
  - lag() returns the value of the previous row.
- **Example**:
-

```
df = df.withColumn("next_value",
F.lead("value").over(window_spec))
```

-

```
df = df.withColumn("previous_value",
F.lag("value").over(window_spec))
```

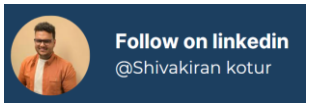
e. Aggregation Functions

Window functions can also be used to compute aggregated values over a specified window.

- **Example for Average:**

```
df = df.withColumn("avg_value", F.avg("value").over(window_spec))
```

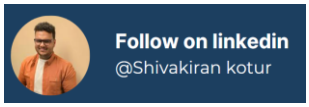
- Other common aggregation functions that can be used include:
  - **Sum**: F.sum("column_name").over(window_spec)
  - **Min**: F.min("column_name").over(window_spec)
  - **Max**: F.max("column_name").over(window_spec)



5. Putting It All Together

Here's a complete example of how to use the various window functions in PySpark:

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Initialize Spark session
spark =
SparkSession.builder.appName("WindowFunctionsExample").getOrCreate(
)

# Sample DataFrame
data = [
    ("A", "X", 1, "2023-01-01"),
    ("A", "X", 2, "2023-01-02"),
    ("A", "Y", 3, "2023-01-01"),
    ("A", "Y", 3, "2023-01-02"),
    ("B", "X", 5, "2023-01-01"),
    ("B", "X", 4, "2023-01-02"),
]
columns = ["category", "sub_category", "value", "timestamp"]
df = spark.createDataFrame(data, columns)

# Define the window specification
window_spec = Window.partitionBy("category",
"sub_category").orderBy(F.col("timestamp"), F.col("value"))

# Apply window functions
df = df.withColumn("row_number", F.row_number().over(window_spec))
df = df.withColumn("rank", F.rank().over(window_spec))
df = df.withColumn("dense_rank", F.dense_rank().over(window_spec))
df = df.withColumn("next_value", F.lead("value").over(window_spec))
df = df.withColumn("previous_value",
F.lag("value").over(window_spec))
df = df.withColumn("avg_value", F.avg("value").over(window_spec))

# Show the results
df.show()
```



category	sub_category	value	timestamp	row_number	rank	dense_rank	next_value	previous_value	avg_value
A	X	1	2023-01-01	1	1	1	2	null	1.0
A	X	2	2023-01-02	2	2	2	null	1	1.5
A	Y	3	2023-01-01	1	1	1	3	null	3.0
A	Y	3	2023-01-02	2	2	2	null	3	3.0
B	X	5	2023-01-01	1	1	1	4	null	5.0
B	X	4	2023-01-02	2	2	2	null	5	4.5

6. Conclusion

Window functions in PySpark are powerful tools for analyzing data within groups while retaining row-level detail. By understanding how to define window specifications and apply various functions, you can perform complex data analyses efficiently.