# What are the uses and advantages of Window Functions in PySpark?

Authored by

**stats writer**

June 24, 2024

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2024). *What are the uses and advantages of Window Functions in PySpark?*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=151199

Window Functions in PySpark are powerful tools that allow for efficient and flexible data manipulation and analysis. They are essentially specialized functions that operate on a subset of data within a larger dataset, known as a window.

One of the main uses of Window Functions is to perform calculations and aggregations on specific groups or partitions of data. This allows for more precise and targeted analysis, as well as the ability to compare data within different groups.

Another advantage of Window Functions is their ability to add new columns to a dataset based on the results of the window calculation. This can be useful for creating rankings, adding cumulative sums, or identifying outliers within a dataset.

Additionally, Window Functions can be used to perform time-based calculations, such as calculating moving averages or identifying trends within a dataset.

Overall, the use of Window Functions in PySpark offers a more efficient and versatile approach to data analysis, allowing for more complex and precise calculations to be performed on large datasets.

PySpark Window functions are used to calculate results, such as the rank, row number, etc., over a range of input rows. In this article, I've explained the concept of window functions, syntax, and finally how to use them with PySpark SQL and PySpark DataFrame API. These are handy when making aggregate operations in a specific window frame on DataFrame columns. You can also create UDF to achieve similar functionality but there is a catch.

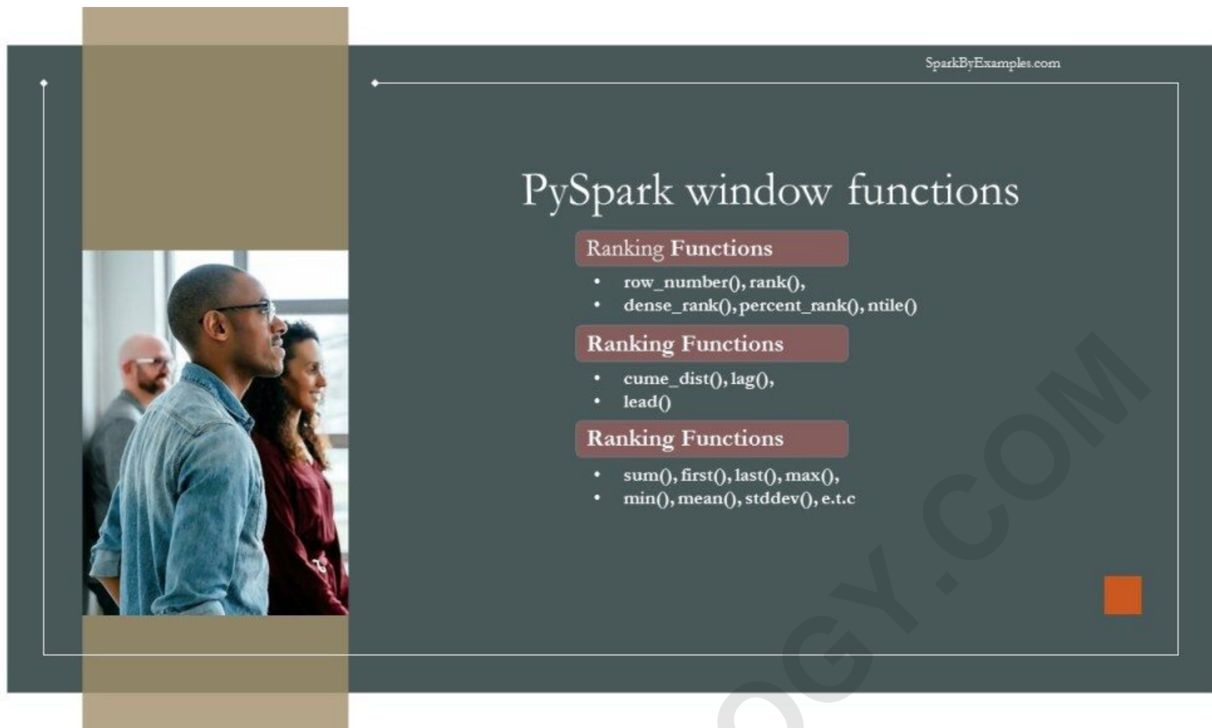
PySpark SQL Functions Explained with Examples

Whenever feasible, consider utilizing standard libraries like window functions as they offer enhanced safety during compile-time, handle null values more effectively, and often deliver better performance compared to user-defined functions (UDFs). If performance is critical for your application, it's advisable to minimize the use of custom UDFs as they do not guarantee optimal performance.

ARABPSYCHOLOGY.COM

## 1. Window Functions

PySpark Window functions operate on a group of rows (like frame, partition) and return a single value for every input row. PySpark SQL supports three kinds of window functions:




PySpark Window Functions

The table below defines Ranking and Analytic functions; for aggregate functions, we can use any existing aggregate functions as a window function.

To operate on a group, first, we need to partition the data using Window.partitionBy() , and for row number and rank function, we need to additionally order by on partition data using `orderBy` clause.

| WINDOW FUNCTIONS USAGE & SYNTAX | PYSPARK WINDOW FUNCTIONS DESCRIPTION |
|---|---|
| row_number() | Returns a sequential number starting from 1 within a window partition |
| rank() | Returns the rank of rows within a window partition, with gaps. |
| percent_rank() | Returns the percentile rank of rows within a window partition. |
| dense_rank() | Returns the rank of rows within a window partition without any gaps. Where as Rank() returns rank with gaps. |
| ntile(n) | Returns the ntile id in a window partition |
| cume_dist() | Returns the cumulative distribution of values within a window partition |
| lag(e, offset)<br>lag(columnname, offset)<br>lag(columnname, offset, defaultvalue) | In PySpark, the lag() function retrieves the value of a column from a preceding row within the same window. It enables users to compare values across adjacent rows and perform calculations based on the difference or relationship between consecutive values in a DataFrame. |

ARABPSYCHOLOGY.COM

| WINDOW FUNCTIONS USAGE & SYNTAX | PYSPARK WINDOW FUNCTIONS DESCRIPTION |
|---|---|
| lead(columnname, offset)<br>lead(columnname, offset)<br>lead(columnname, offset, defaultvalue) | The lead() function in PySpark retrieves the value of a column from a succeeding row within the same window. It enables users to access values ahead of the current row and perform comparisons or calculations based on future values in a DataFrame. |

PySpark Window Functions

Before we start with an example, let's create a PySpark DataFrame to work with.

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

simpleData = (("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)
)

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

Yields below output

```
# Output:
root
|-- employee_name: string (nullable = true)
|-- department: string (nullable = true)
|-- salary: long (nullable = true)
```

+-------------+----------+------+

ARABPSYCHOLOGY.COM

|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 |
+-------------+----------+-------+

## 2. PySpark Window Ranking functions

PySpark's Window Ranking functions, like `row_number()`, `rank()`, and `dense_rank()`, assign sequential numbers to DataFrame rows based on specified criteria within defined partitions. These functions enable sorting and ranking operations, identifying row positions in partitions based on specific orderings.

The `row_number()` assigns unique sequential numbers, `rank()` provides the ranking with gaps, and `dense_rank()` offers ranking without gaps. They're valuable in selecting top elements within groups and bottom elements within groups, facilitating analysis of data distributions, and identifying the highest or lowest values within partitions in PySpark DataFrames.

### 2.1 row_number Window Function

`row_number()` window function gives the sequential row number starting from 1 to the result of each window partition.

```
# row_number() example
from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec = Window.partitionBy("department").orderBy("salary")
```

df.withColumn("row_number",row_number().over(windowSpec))
.show(truncate=False)

ARABPSYCHOLOGY.COM

Yields below output.

```
# Output:
+-------------+----------+------+----------+
|employee_name|department|salary|row_number|
+-------------+----------+------+----------+
|James |Sales |3000 |1 |
|James |Sales |3000 |2 |
|Robert |Sales |4100 |3 |
|Saif |Sales |4100 |4 |
|Michael |Sales |4600 |5 |
|Maria |Finance |3000 |1 |
|Scott |Finance |3300 |2 |
|Jen |Finance |3900 |3 |
|Kumar |Marketing |2000 |1 |
|Jeff |Marketing |3000 |2 |
+-------------+----------+------+----------+
```

## 2.2 rank Window Function

`rank()` window function provides a rank to the result within a window partition. This function leaves gaps in rank when there are ties.

```
# rank() example
from pyspark.sql.functions import rank
df.withColumn("rank",rank().over(windowSpec))
.show()
```

Yields below output.

```
# Output:
+-------------+----------+------+----+
|employee_name|department|salary|rank|
+-------------+----------+------+----+
| James| Sales| 3000| 1|
| James| Sales| 3000| 1|
| Robert| Sales| 4100| 3|
| Saif| Sales| 4100| 3|
| Michael| Sales| 4600| 5|
```

ARABPSYCHOLOGY.COM

```
| Maria| Finance| 3000| 1|
| Scott| Finance| 3300| 2|
| Jen| Finance| 3900| 3|
| Kumar| Marketing| 2000| 1|
| Jeff| Marketing| 3000| 2|
+-------------+----------+------+----+
```

This is the same as the `RANK` function in SQL.

## 2.3 dense_rank Window Function

`dense_rank()` window function is used to get the result with rank of rows within a window partition without any gaps. This is similar to `rank()` function difference being rank function leaves gaps in rank when there are ties.

```
# dense_rank() example
from pyspark.sql.functions import dense_rank
df.withColumn("dense_rank",dense_rank().over(windowSpec))
.show()
```

Yields below output.

```
# Output:
+-------------+----------+------+----------+
|employee_name|department|salary|dense_rank|
+-------------+----------+------+----------+
| James| Sales| 3000| 1|
| James| Sales| 3000| 1|
| Robert| Sales| 4100| 2|
| Saif| Sales| 4100| 2|
| Michael| Sales| 4600| 3|
| Maria| Finance| 3000| 1|
| Scott| Finance| 3300| 2|
| Jen| Finance| 3900| 3|
| Kumar| Marketing| 2000| 1|
| Jeff| Marketing| 3000| 2|
+-------------+----------+------+----------+
```

This is the same as the `DENSE_RANK` function in SQL.

ARABPSYCHOLOGY.COM

## 2.4 percent_rank Window Function

```
# percent_rank() Example
from pyspark.sql.functions import percent_rank
df.withColumn("percent_rank",percent_rank().over(windowSpec))
.show()
```

Yields below output.

```
# Output:
+-------------+----------+------+------------+
|employee_name|department|salary|percent_rank|
+-------------+----------+------+------------+
| James| Sales| 3000| 0.0|
| James| Sales| 3000| 0.0|
| Robert| Sales| 4100| 0.5|
| Saif| Sales| 4100| 0.5|
| Michael| Sales| 4600| 1.0|
| Maria| Finance| 3000| 0.0|
| Scott| Finance| 3300| 0.5|
| Jen| Finance| 3900| 1.0|
| Kumar| Marketing| 2000| 0.0|
| Jeff| Marketing| 3000| 1.0|
+-------------+----------+------+------------+
```

This is the same as the `PERCENT_RANK` function in SQL.

ARABPSYCHOLOGY.COM

## 2.5 ntile Window Function

`ntile()` window function returns the relative rank of result rows within a window partition. In the below example we have used 2 as an argument to ntile hence it returns ranking between 2 values (1 and 2)

```
#ntile() Example
from pyspark.sql.functions import ntile
df.withColumn("ntile",ntile(2).over(windowSpec))
.show()
```

Yields below output.

```
# Output:
+-------------+----------+------+-----+
|employee_name|department|salary|ntile|
+-------------+----------+------+-----+
| James| Sales| 3000| 1|
| James| Sales| 3000| 1|
| Robert| Sales| 4100| 1|
| Saif| Sales| 4100| 2|
| Michael| Sales| 4600| 2|
| Maria| Finance| 3000| 1|
| Scott| Finance| 3300| 1|
| Jen| Finance| 3900| 2|
| Kumar| Marketing| 2000| 1|
| Jeff| Marketing| 3000| 2|
+-------------+----------+------+-----+
```

This is the same as the `NTILE` function in SQL.

## 3. PySpark Window Analytic Functions

### 3.1 cume_dist Window Function

**`cume_dist()`:** This function computes the cumulative distribution of a value within a window partition. It calculates the relative rank of a value within the partition. The result ranges from 0 to 1, where a value of 0 indicates the lowest value in the partition, and 1 indicates the highest. It's useful for understanding the distribution of values compared to others within the same partition.

This is the same as the `DENSE_RANK` function in SQL.

```
# cume_dist() Example
from pyspark.sql.functions import cume_dist
df.withColumn("cume_dist",cume_dist().over(windowSpec))
.show()
```

```
# Output:
+-------------+----------+------+------------------+
|employee_name|department|salary| cume_dist|
+-------------+----------+------+------------------+
| James| Sales| 3000| 0.4|
| James| Sales| 3000| 0.4|
```

ARABPSYCHOLOGY.COM

```
| Robert| Sales| 4100| 0.8|
| Saif| Sales| 4100| 0.8|
| Michael| Sales| 4600| 1.0|
| Maria| Finance| 3000|0.3333333333333333|
| Scott| Finance| 3300|0.6666666666666666|
| Jen| Finance| 3900| 1.0|
| Kumar| Marketing| 2000| 0.5|
| Jeff| Marketing| 3000| 1.0|
+-------------+----------+------+------------------+
```

## 3.2 lag Window Function

This is the same as the `LAG` function in SQL. The `lag()` function allows you to access a previous row's value within the partition based on a specified offset. It retrieves the column value from the previous row, which can be helpful for comparative analysis or calculating differences between consecutive rows.

```
# lag() Example
from pyspark.sql.functions import lag
df.withColumn("lag",lag("salary",2).over(windowSpec))
.show()
```

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

ARABPSYCHOLOGY.COM

### 3.3 lead Window Function

This is the same as the `LEAD` function in SQL. Similar to `lag()`, the `lead()` function retrieves the column value from the following row within the partition based on a specified offset. It helps in accessing subsequent row values for comparison or predictive analysis.

```
# lead() Example
from pyspark.sql.functions import lead
df.withColumn("lead",lead("salary",2).over(windowSpec))
.show()
```

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

ARABPSYCHOLOGY.COM

## 4. PySpark Window Aggregate Functions

PySpark's window aggregate functions, such as `sum()`, `avg()`, and `min()`, compute aggregated values within specified window partitions. These functions perform calculations across rows related to each row in the partition, enabling cumulative or comparative analyses within specific groupings.

They allow computation of running totals, averages, minimums, or other aggregations over defined windows, assisting in tasks like calculating moving averages, cumulative sums, or identifying extreme values within subsets of data in PySpark DataFrames. These functions offer insights into data trends, patterns, and statistical summaries within specified groupings or orderings.

In this section, I will explain how to calculate sum, min, max for each department using PySpark SQL Aggregate window functions and `WindowSpec`. When working with Aggregate functions, we

don't need to use order by clause.

```
# Aggregate functions examples
windowSpecAgg = Window.partitionBy("department")
from pyspark.sql.functions import col,avg,sum,min,max,row_number
df.withColumn("row",row_number().over(windowSpec))
.withColumn("avg", avg(col("salary")).over(windowSpecAgg))
.withColumn("sum", sum(col("salary")).over(windowSpecAgg))
.withColumn("min", min(col("salary")).over(windowSpecAgg))
.withColumn("max", max(col("salary")).over(windowSpecAgg))
.where(col("row")==1).select("department","avg","sum","min","max")
.show()
```

This yields below output

```
# Output:
+----------+------+-----+----+-----+
|department| avg| sum| min| max|
+----------+------+-----+----+-----+
| Sales|3760.0|18800|3000|4600|
| Finance|3400.0|10200|3000|3900|
| Marketing|2500.0| 5000|2000|3000|
+----------+------+-----+----+-----+
```

## Source Code of Window Functions Example

```
import pyspark
from pyspark.sql import SparkSession
```

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

simpleData = (("James", "Sales", 3000),
("Michael", "Sales", 4600),
("Robert", "Sales", 4100),
("Maria", "Finance", 3000),
("James", "Sales", 3000),
("Scott", "Finance", 3300),
("Jen", "Finance", 3900),
("Jeff", "Marketing", 3000),

ARABPSYCHOLOGY.COM

```
("Kumar", "Marketing", 2000),
("Saif", "Sales", 4100)
)

columns=

df = spark.createDataFrame(data = simpleData, schema = columns)

df.printSchema()
df.show(truncate=False)

from pyspark.sql.window import Window
from pyspark.sql.functions import row_number
windowSpec = Window.partitionBy("department").orderBy("salary")

df.withColumn("row_number",row_number().over(windowSpec))
.show(truncate=False)

from pyspark.sql.functions import rank
df.withColumn("rank",rank().over(windowSpec))
.show()

from pyspark.sql.functions import dense_rank
df.withColumn("dense_rank",dense_rank().over(windowSpec))
.show()

from pyspark.sql.functions import percent_rank
df.withColumn("percent_rank",percent_rank().over(windowSpec))
.show()

from pyspark.sql.functions import ntile
df.withColumn("ntile",ntile(2).over(windowSpec))
.show()

from pyspark.sql.functions import cume_dist
df.withColumn("cume_dist",cume_dist().over(windowSpec))
.show()

from pyspark.sql.functions import lag
df.withColumn("lag",lag("salary",2).over(windowSpec))
.show()

from pyspark.sql.functions import lead
```

ARABPSYCHOLOGY.COM

```
df.withColumn("lead",lead("salary",2).over(windowSpec))
.show()

windowSpecAgg = Window.partitionBy("department")
from pyspark.sql.functions import col,avg,sum,min,max,row_number
df.withColumn("row",row_number().over(windowSpec))
.withColumn("avg", avg(col("salary")).over(windowSpecAgg))
.withColumn("sum", sum(col("salary")).over(windowSpecAgg))
.withColumn("min", min(col("salary")).over(windowSpecAgg))
.withColumn("max", max(col("salary")).over(windowSpecAgg))
.where(col("row")==1).select("department","avg","sum","min","max")
.show()
```

The complete source code is available at PySpark Examples GitHub for reference.

## Conclusion

In this tutorial, you have learned what PySpark SQL Window functions, their syntax, and how to use them with aggregate functions, along with several examples in Scala.

## Related Articles

## References

In conclusion, PySpark Window functions are analytical functions that operate on a subset of rows, known as a window, within a larger result set. They are essential for performing advanced data analysis tasks, such as calculating running totals, rankings, moving averages, and aggregations, while retaining the context of individual rows.

Window functions enable partitioning, ordering, and framing of data within the window, allowing for precise control over the computation of results. By providing insights into data trends, patterns, and relationships across different segments, window functions empower analysts and data scientists to derive deeper insights and make informed decisions from their datasets.

Happy Learning !!

ARABPSYCHOLOGY.COM