Window functions:

→ Window functions perform calculations across a set of related rows without collapsing rows.

> groupBy() → reduces rows
> window functions → keeps all rows, just add extra info.

over() clause:

↓

> tells spark how a window function should look at other rows.
> without over();
a window function does not know:
- which rows to consider
- how to group them
- how to order them.

Basic Syntax:

`window_function().over(window_spec)`

Ex: `row_number().over(Window.partitionBy("Col1").orderBy("Col2"))`

⇒ window functions;
↓ have Aggregate functions,
rank functions,
value/Analytics functions

window_function().over(window_spec)

⇒ window_functions

- rank functions
- value | Analytics functions
- Aggregate functions

① Rank functions:

- row_Number()
- rank()
- dense_rank()
- percent_rank()
- ntile(n)
- cume_dist()

② value | Analytics functions

- lag()
- lead()
- first_value()
- last_value()
- nth_value()

③ Aggregate functions

- sum()
- avg()
- min()
- max()
- count()
- stddev()
- varience()

window syntax

over clause

- window Function
- Partition clause
- order clause
- frame clause

{we can use one, or two, or all}

PartitionBy():

> Divides data into groups

> Each group is processed independently

OrderBy():

> Defines the order of rows inside each partition

> required for ranking, lag, lead, running totals

Frame clause:

- rowsBetween()
  (or)
- rangeBetween()

Purpose: controls how many rows are considered relative to the current row.

```
over(window.partitionBy(....)
            .orderBy(....)
            .rowsBetween(....))
```

⇒ We can use:

- > only partition
- > partition + order
- > partition + order + frame
- > only order
- > (or) none (global window)

Window Aggregate Functions

using only over() {empty clase}

Syntax: `sum(col1).over()`

⇒ Sample data:

emp	dept	salary
A	IT	1000
B	IT	1500
C	HR	1200
D	HR	900

sum()
avg()
count()
min()
max()
stddev()
variences()

These are Normal aggregate functions; but used with over()

---

① sum():

```
df.withColumn(
        "total_salary",
        sum("salary").over()
)
```

- Adds all salaries
- repeats the results on every row

o/p:

emp	dept	salary	total_salary
A	IT	1000	4600
B	IT	1500	4600
C	HR	1200	4600
D	HR	900	4600

② avg():

```
df.withColumn(
    "avg_salary",
    avg("salary").over()
)
```

o/p

emp	salary	avg_salary
A	1000	1150
B	1500	1150
C	1200	1150
D	900	1150

③ count():

```
df.withColumn(
    "total_rows",
    count('*').over()
)
```

o/p

emp	total_rows
A	4
B	4
C	4
D	4

④ min():

```
df.withColumn(
    "min_salary",
    min("salary").over()
)
```

⑤ max():

```
df.withColumn(
    "max_salary",
    max("salary").over()
)
```

⑥ stddev():

```
df.withColumn(
    "salary_stddev",
    stddev("salary").over()
)
```

⑦ variance():

```
df.withColumn(
    "salary_variance",
    variance('salary').over()
)
```

* whole table treated as one window
* result repeated for every row

using empty `over()` means:

* whole DataFrame is one window
* same result on every row

using partitionBy() & orderBy() in the over() clause

Syntax:

```
sum('col1').over( Window.partitionBy('col2')
                                .orderBy('col3')
                )
```

Sample data:

emp	dept	salary
A	IT	1000
B	IT	1500
C	IT	1200
D	HR	900
E	HR	1100

Note:

When we use:

over(partitionBy(...).orderBy(...))

spark automatically applies this frame:

Rows Between unBounded Preceding and current Row

① sum():

```
from pyspark.sql.window import Window

df.withColumn(
        "running_salary",
        Sum("salary").over(
                , Window.partitionBy('dept')
                        .orderBy('salary')
                )
)
```

emp	dept	salary	running_salary
A	IT	1000	1000
C	IT	1200	2200
B	IT	1500	3700
D	HR	900	900
E	HR	1100	2000

Due to Default Row between behaviors.

② avg():

```
df.withColumn(
    "running_avg_salary",
    avg('salary').over(
        Window.partitionBy('dept').orderBy('salary')
    )
)
```

③ min():

```
df.withColumn(
    "running_min_so_far",
    min('salary').over(
        Window.partitionBy('dept')
              .orderBy('salary')
    )
)
```

④ max():

```
df.withColumn(
    "running_max_so_far",
    max('salary').over(
        Window.partitionBy('dept')
              .orderBy('salary')
    )
)
```

emp	dept	salary	running_avg_salary	running_min	running_max
A	IT	1000	1000.00	1000	1000
C	IT	1200	1100.00	1000	1200
B	IT	1500	1233.33	1000	1500
D	HR	900	900.00	900	900
E	HR	1100	1000.00	900	1100

⑤ count()

```
df.withColumn(
        "running_count",
        count("*").over(
                Window.partitionBy('dept')
                      .orderBy('salary')
        )
)
```

Emp	dept	Salary	running_count
A	IT	1000	1
C	IT	1200	2
B	IT	1500	3
D	HR	900	1
E	HR	1100	2

⑥ stddev():

```
df.withColumn(
        'salary_stddev',
        stddev('salary').over(
                Window.partitionBy('dept').orderBy('salary')
        )
```

⑦ variance():

```
df.withColumn(
        'salary_variance', variance('salary').over(
                Window.partitionBy('dept').orderBy('salary'))
```

Emp	dept	Salary	salary_stddev	salary_variance
A	IT	1000	NULL	NULL
C	IT	1200	141.42	20000
B	IT	1500	251.66	63333
D	HR	900	NULL	NULL
E	HR	1100	141.42	20000