How can I pivot and unpivot a DataFrame in PySpark?

Authored by

stats writer

June 24, 2024

Pivoting and unpivoting a DataFrame in PySpark refers to the process of transforming the data structure from a long format to a wide format and vice versa. This allows for better organization and analysis of data, as well as facilitating the creation of different visualizations. Pivoting involves converting rows into columns, while unpivoting involves converting columns back into rows. This can be achieved in PySpark by using the pivot and unpivot functions, which allow for the manipulation of data based on specific columns and values. By understanding and utilizing these functions, users can effectively reshape their data and gain valuable insights from their PySpark DataFrames.

PySpark `pivot()` function is used to rotate/transpose the data from one column into multiple Dataframe columns and back using unpivot(). Pivot() It is an aggregation where one of the grouping columns values is transposed into individual columns with distinct data.

Syntax

```
pivot_df = original_df.groupBy("grouping_column").pivot("pivot_column").agg({"agg_column": "agg_function"})
```

grouping_column: The column used for grouping.

pivot_column: The column whose distinct values become new columns.

agg_column: The column for which aggregation is applied (e.g., using a function like `sum`, `avg`, etc.).

This tutorial describes and provides a PySpark example on how to create a Pivot table on DataFrame and Unpivot back.

Let's create a PySpark DataFrame to work with.

```
# Imports
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import expr
#Create spark session
data =
```

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

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

DataFrame 'df' consists of 3 columns `Product`, `Amount`, and `Country` as shown below.

```
# Output
root
|-- Product: string (nullable = true)
|-- Amount: long (nullable = true)
|-- Country: string (nullable = true)
```

```
+-------+------+-------+
|Product|Amount|Country|
+-------+------+-------+
|Banana |1000 |USA |
|Carrots|1500 |USA |
|Beans |1600 |USA |
|Orange |2000 |USA |
|Orange |2000 |USA |
|Banana |400 |China |
|Carrots|1200 |China |
|Beans |1500 |China |
|Orange |4000 |China |
|Banana |2000 |Canada |
|Carrots|2000 |Canada |
|Beans |2000 |Mexico |
+-------+------+-------+
```

Pivot PySpark DataFrame

PySpark SQL provides `pivot()` function to rotate the data from one column into multiple columns. It is an aggregation where one of the grouping columns values is transposed into individual columns with distinct data. To get the total amount exported to each country of each product, will do group by `Product`, pivot by `Country`, and the sum of `Amount`.

```
# Applying pivot()
pivotDF = df.groupBy("Product").pivot("Country").sum("Amount")
pivotDF.printSchema()
```

```
pivotDF.show(truncate=False)
```

This will transpose the countries from DataFrame rows into columns and produces the below output. where ever data is not present, it represents as `null` by default.

```
# Output
root
|-- Product: string (nullable = true)
|-- Canada: long (nullable = true)
|-- China: long (nullable = true)
|-- Mexico: long (nullable = true)
|-- USA: long (nullable = true)

+-------+------+-----+------+----+
|Product|Canada|China|Mexico|USA |
+-------+------+-----+------+----+
|Orange |null |4000 |null |4000|
|Beans |null |1500 |2000 |1600|
|Banana |2000 |400 |null |1000|
|Carrots|2000 |1200 |null |1500|
+-------+------+-----+------+----+
```

Pivot Performance improvement in PySpark 2.0

version 2.0 on-wards performance has been improved on Pivot, however, if you are using the lower version; note that pivot is a very expensive operation hence, it is recommended to provide column data (if known) as an argument to function as shown below.

```
countries =
pivotDF = df.groupBy("Product").pivot("Country", countries).sum("Amount")
pivotDF.show(truncate=False)
```

Another approach is to do two-phase aggregation. PySpark 2.0 uses this implementation in order to improve the performance Spark-13749

```
pivotDF = df.groupBy("Product","Country")
.sum("Amount")
.groupBy("Product")
```

```
.pivot("Country")
.sum("sum(Amount)")
pivotDF.show(truncate=False)
```

The above two examples return the same output but with better performance.

Unpivot PySpark DataFrame

Unpivot is a reverse operation, we can achieve by rotating column values into rows values. PySpark SQL doesn't have unpivot function hence will use the `stack()` function. Below code converts column countries to row.

```
# Applying unpivot()
from pyspark.sql.functions import expr
unpivotExpr = "stack(3, 'Canada', Canada, 'China', China, 'Mexico', Mexico) as (Country,Total)"
unPivotDF = pivotDF.select("Product", expr(unpivotExpr))
.where("Total is not null")
unPivotDF.show(truncate=False)
unPivotDF.show()
```

It converts pivoted column "country" to rows.

```
# Output
+-------+-------+-----+
|Product|Country|Total|
+-------+-------+-----+
| Orange| China| 4000|
| Beans| China| 1500|
| Beans| Mexico| 2000|
| Banana| Canada| 2000|
| Banana| China| 400|
|Carrots| Canada| 2000|
|Carrots| China| 1200|
+-------+-------+-----+
```

Complete Example

The complete code can be downloaded from GitHub

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import expr
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
```

data =

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

pivotDF = df.groupBy("Product").pivot("Country").sum("Amount")
pivotDF.printSchema()
pivotDF.show(truncate=False)

pivotDF = df.groupBy("Product","Country")
.sum("Amount")
.groupBy("Product")
.pivot("Country")
.sum("sum(Amount)")
pivotDF.printSchema()
pivotDF.show(truncate=False)

""" unpivot """
unpivotExpr = "stack(3, 'Canada', Canada, 'China', China, 'Mexico', Mexico) as (Country,Total)"
unPivotDF = pivotDF.select("Product", expr(unpivotExpr))
.where("Total is not null")
unPivotDF.show(truncate=False)

Frequently Asked Questions on pivot() and unpivot()

Can we do PySpark DataFrame transpose or pivot without aggregation?

Of course you can, but unfortunately, you can't achieve using the Pivot function. However, pivoting or transposing the DataFrame structure without aggregation from rows to columns and columns to rows can be easily done using PySpark and Scala hack. please refer to stackoverflow example.

What happens if there are duplicate entries for the pivot column?

The `pivot` operation requires unique combinations of grouping and pivot columns. If there are

duplicate entries, you may need to perform an aggregation (e.g., using `agg` parameter) to resolve the conflict.

How does the `pivot` operation handle missing values?

The `pivot` operation fills missing values with null. If you need to handle missing values differently, you can use methods like `fillna` or `na.drop` on the pivoted DataFrame.

Conclusion:

We have seen how to Pivot DataFrame with PySpark example and Unpivot it back using SQL functions. And also saw how PySpark 2.0 changes have improved performance by doing two-phase aggregation.

Happy Learning !!

Related Articles