How can the PySpark withColumn() function be used and what are some examples of its usage?

Authored by

stats writer

June 24, 2024

The PySpark withColumn() function is a powerful tool used in data processing and manipulation within the PySpark framework. It allows users to add a new column or modify an existing column in a dataframe by applying a user-defined function to each row. This function takes in two parameters: the name of the new column and the function to be applied. The new column is then added to the dataframe with the specified name, containing the results of the function applied to each row. This function is extremely useful in data transformation tasks such as data cleaning, feature engineering, and data aggregation.

Some examples of using the PySpark withColumn() function include adding a new column that calculates the total sales by multiplying the quantity and price columns, converting a column from string to integer data type, and creating a new column that categorizes data based on a certain condition. Additionally, the withColumn() function can also be used to rename existing columns, drop columns, and perform various mathematical and statistical operations on columns. Overall, the PySpark withColumn() function is an essential tool for data manipulation and can greatly enhance the efficiency and accuracy of data analysis tasks in the PySpark environment.

PySpark `withColumn()` is a transformation function of DataFrame which is used to change the value, convert the datatype of an existing column, create a new column, and many more. In this post, I will walk you through commonly used PySpark DataFrame column operations using withColumn() examples.

First, let's create a DataFrame to work with.

```
data =

columns =
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('SparkByExamples.com').getOrCreate()
df = spark.createDataFrame(data=data, schema = columns)
```

1. Change DataType using PySpark withColumn()

By using PySpark `withColumn()` on a DataFrame, we can cast or change the data type of a column. In order to change data type, you would also need to use `cast()` function along with withColumn(). The below statement changes the datatype from `String` to `Integer` for the `salary` column.

```
df.withColumn("salary",col("salary").cast("Integer")).show()
```

2. Update The Value of an Existing Column

PySpark `withColumn()` function of DataFrame can also be used to change the value of an existing column. In order to change the value, pass an existing column name as a first argument and a value to be assigned as a second argument to the withColumn() function. Note that the second argument should be `Column` type . Also, see Different Ways to Update PySpark DataFrame Column.

```
df.withColumn("salary",col("salary")*100).show()
```

This snippet multiplies the value of "salary" with 100 and updates the value back to "salary" column.

3. Create a Column from an Existing

To add/create a new column, specify the first argument with a name you want your new column to be and use the second argument to assign a value by applying an operation on an existing column. Also, see Different Ways to Add New Column to PySpark DataFrame.

```
df.withColumn("CopiedColumn",col("salary")* -1).show()
```

This snippet creates a new column CopiedColumn" by multiplying "salary" column with value -1.

4. Add a New Column using withColumn()

In order to create a new column, pass the column name you wanted to the first argument of `withColumn()` transformation function. Make sure this new column not already present on DataFrame, if it presents it updates the value of that column.

On below snippet, `PySpark lit()` function is used to add a constant value to a DataFrame column. We can also chain in order to add multiple columns.

```
df.withColumn("Country", lit("USA")).show()
df.withColumn("Country", lit("USA"))
.withColumn("anotherColumn",lit("anotherValue"))
.show()
```

5. Rename Column Name

Though you cannot rename a column using withColumn, still I wanted to cover this as renaming is one of the common operations we perform on DataFrame. To rename an existing column use `withColumnRenamed()` function on DataFrame.

```
df.withColumnRenamed("gender","sex")
.show(truncate=False)
```

6. Drop Column From PySpark DataFrame

Use "drop" function to drop a specific column from the DataFrame.

```
df.drop("salary")
.show()
```

Note: Note that all of these functions return the new DataFrame after applying the functions instead of updating DataFrame.

7. PySpark withColumn() Complete Example

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField, StringType,IntegerType
```

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

data =

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

df2 = df.withColumn("salary",col("salary").cast("Integer"))
df2.printSchema()
df2.show(truncate=False)

```
df3 = df.withColumn("salary",col("salary")*100)
df3.printSchema()
df3.show(truncate=False)

df4 = df.withColumn("CopiedColumn",col("salary")* -1)
df4.printSchema()

df5 = df.withColumn("Country", lit("USA"))
df5.printSchema()

df6 = df.withColumn("Country", lit("USA"))
.withColumn("anotherColumn",lit("anotherValue"))
df6.printSchema()

df.withColumnRenamed("gender","sex")
.show(truncate=False)

df4.drop("CopiedColumn")
.show(truncate=False)
```

The complete code can be downloaded from PySpark withColumn GitHub project

Happy Learning !!

Related Articles