PySpark DataFrame Column Operations

ddas.tech/pyspark-dataframe-column-operations/

November 23, 2022

Created By: Debasis Das (23-Nov-2022) – In this post we will create simple sample codes to explore different **PySpark DataFrame Column Operations** and behavior for a given Spark DataFrame. Column operations are crucial to any spark application development to extract new data features, modify content and filter records based on column values.

```
df("columnName") // On a specific `df` DataFrame.
col("columnName") // A generic column not yet associated with a DataFrame.
col("columnName.field") // Extracting a struct field
```

Table of Contents

Sample Spark DataFrame

Lets start the exercise by creating the spark session and some sample data, In the given sample we have created a Spark DataFrame from Pandas DataFrame.

```
import findspark
findspark.init()

import pyspark
from pyspark.sql import SparkSession
from pyspark import SparkConf
from pyspark.sql.types import StructType, StructField, DateType, StringType,
IntegerType
from pyspark.sql.functions import expr
from pyspark.sql import functions as F

conf = SparkConf().setMaster("local[3]").setAppName("SparkColumnOperations")
spark = SparkSession.builder.config(conf=conf).getOrCreate()
spark.sparkContext.setLogLevel("WARN")

import pandas as pd
import numpy as np

countries = ["USA","Mexico","Brazil","Canada"]
cars = ["BMW X5","BMW X7","Ford Explorer","Ford Expedition","Jeep Wrangler","Jeep
Cherokee"]
weeks = []
for i in range(1,5):
    weeks.append(f"Week_{i}")

num_records = 96
df = pd.DataFrame({"country":np.random.choice(countries,num_records),
                   "car":np.random.choice(cars,num_records),
                   "week":np.random.choice(weeks,num_records),
                   "units_sales":np.random.randint(20,size = num_records),
                   "used_new":np.random.choice(["Used","New"],num_records),
                   "price_per_unit":np.random.randint(low = 20000, high = 55000,size
= num_records)
                  })
print(df.head(5))
print(df.shape)
```

```
  country              car    week  units_sales used_new  price_per_unit
0  Mexico    Jeep Wrangler  Week_2            3     Used           21005
1  Canada            BMW X7  Week_1            5      New           22531
2     USA    Jeep Wrangler  Week_3            5      New           45463
3  Brazil  Ford Expedition  Week_1           18     Used           47425
4  Brazil    Jeep Cherokee  Week_1            9      New           40574
(96, 6)
```

Creating a Spark DataFrame from Pandas DataFrame

```
sDF=spark.createDataFrame(df)
sDF.printSchema()
sDF.show(5)

root
 |-- country: string (nullable = true)
 |-- car: string (nullable = true)
 |-- week: string (nullable = true)
 |-- units_sales: long (nullable = true)
 |-- used_new: string (nullable = true)
 |-- price_per_unit: long (nullable = true)

+-------+---------------+------+-----------+--------+--------------+
|country|            car|  week|units_sales|used_new|price_per_unit|
+-------+---------------+------+-----------+--------+--------------+
|    USA|  Jeep Wrangler|Week_2|          7|     New|         41790|
| Canada|  Ford Explorer|Week_3|          9|    Used|         46232|
| Canada|  Jeep Cherokee|Week_2|         10|    Used|         25325|
| Mexico|Ford Expedition|Week_1|          2|    Used|         54640|
|    USA|  Jeep Cherokee|Week_4|         19|     New|         43053|
+-------+---------------+------+-----------+--------+--------------+
only showing top 5 rows
```

Creating a new column in the Spark DataFrame

In the below code we have created a new column called Revenue and initialized the same with 0 values

```
sDF = sDF.withColumn("revenue", F.lit(0))
sDF.show(5)

+-------+---------------+------+-----------+--------+--------------+-------+
|country|            car|  week|units_sales|used_new|price_per_unit|revenue|
+-------+---------------+------+-----------+--------+--------------+-------+
|    USA|  Jeep Wrangler|Week_2|          7|     New|         41790|      0|
| Canada|  Ford Explorer|Week_3|          9|    Used|         46232|      0|
| Canada|  Jeep Cherokee|Week_2|         10|    Used|         25325|      0|
| Mexico|Ford Expedition|Week_1|          2|    Used|         54640|      0|
|    USA|  Jeep Cherokee|Week_4|         19|     New|         43053|      0|
+-------+---------------+------+-----------+--------+--------------+-------+
only showing top 5 rows
```

Update Value of a column based on calculation on other column values

```
sDF = sDF.withColumn('revenue', (sDF.units_sales * sDF.price_per_unit))
sDF.show(5)
```

```
+-------+---------------+------+-----------+--------+-------------+-------+
|country|            car|  week|units_sales|used_new|price_per_unit|revenue|
+-------+---------------+------+-----------+--------+-------------+-------+
|    USA|  Jeep Wrangler|Week_2|          7|     New|         41790| 292530|
| Canada|  Ford Explorer|Week_3|          9|    Used|         46232| 416088|
| Canada|  Jeep Cherokee|Week_2|         10|    Used|         25325| 253250|
| Mexico|Ford Expedition|Week_1|          2|    Used|         54640| 109280|
|    USA|  Jeep Cherokee|Week_4|         19|     New|         43053| 818007|
+-------+---------------+------+-----------+--------+-------------+-------+
only showing top 5 rows
```

Update Value of Column based on calculation on same column

In the below sample we are increasing the revenue by 10%

```
sDF = sDF.withColumn('revenue', (sDF.revenue *1.1))
sDF.select(sDF.country,sDF.car, sDF.week, sDF.revenue).show(5)
```

```
+-------+---------------+------+------------------+
|country|            car|  week|           revenue|
+-------+---------------+------+------------------+
| Mexico|  Jeep Wrangler|Week_2| 76248.15000000001|
| Canada|         BMW X7|Week_1|136312.55000000002|
|    USA|  Jeep Wrangler|Week_3| 275051.1500000001|
| Brazil|Ford Expedition|Week_1|1032916.5000000002|
| Brazil|  Jeep Cherokee|Week_1| 441850.8600000001|
+-------+---------------+------+------------------+
only showing top 5 rows
```

Renaming a Spark DataFrame Column

```
sDF = sDF.withColumnRenamed("revenue", "sales_revenue")
```

Check if Column exists in a DataFrame

```
listOfColumns = sDF.columns
print(listOfColumns)
#['country', 'car', 'week', 'units_sales', 'used_new', 'price_per_unit',
'sales_revenue']

print("country".upper() in (name.upper() for name in sDF.columns))
# True

print(sDF.schema.simpleString())

#struct<country:string,car:string,week:string,units_sales:bigint,used_new:string,price

print(sDF.schema.fieldNames())

['country', 'car', 'week', 'units_sales', 'used_new', 'price_per_unit',
'sales_revenue']
```

Select one or more columns

```
sDF1 = sDF.select("country","car","week","sales_revenue")
sDF1.show(5)
```

```
+-------+---------------+------+------------------+
|country|            car|  week|     sales_revenue|
+-------+---------------+------+------------------+
| Mexico|  Jeep Wrangler|Week_2| 76248.15000000001|
| Canada|         BMW X7|Week_1|136312.55000000002|
|    USA|  Jeep Wrangler|Week_3| 275051.1500000001|
| Brazil|Ford Expedition|Week_1|1032916.5000000002|
| Brazil|  Jeep Cherokee|Week_1| 441850.8600000001|
+-------+---------------+------+------------------+
only showing top 5 rows
```

Unique Values in a Column

```
sDF2 = sDF.select("country").distinct()
sDF2.show()
```

```
+-------+
|country|
+-------+
|    USA|
| Mexico|
| Canada|
| Brazil|
+-------+
```

Drop one or more columns

```
sDF3 = sDF.drop("sales_revenue","used_new")
print("Showing DataFrame with Dropped Columns")
sDF3.show(5)
```

```
Showing DataFrame with Dropped Columns
+-------+---------------+------+-----------+--------------+
|country|            car|  week|units_sales|price_per_unit|
+-------+---------------+------+-----------+--------------+
| Mexico|  Jeep Wrangler|Week_2|          3|         21005|
| Canada|         BMW X7|Week_1|          5|         22531|
|    USA|  Jeep Wrangler|Week_3|          5|         45463|
| Brazil|Ford Expedition|Week_1|         18|         47425|
| Brazil|  Jeep Cherokee|Week_1|          9|         40574|
+-------+---------------+------+-----------+--------------+
only showing top 5 rows
```

Column Operations

```
data=[("Physics",89.3,93.2),
      ("Chemistry",88.2,76.5),
      ("Biology",76.3,88.1)]
sdf4 =spark.createDataFrame(data).toDF("Subject","Mid_Term","Finals")
sdf4.show()

+---------+--------+------+
|  Subject|Mid_Term|Finals|
+---------+--------+------+
|  Physics|    89.3|  93.2|
|Chemistry|    88.2|  76.5|
|  Biology|    76.3|  88.1|
+---------+--------+------+

sdf4.select((sdf4.Mid_Term + sdf4.Finals)/2).show()

+-------------------------+
|((Mid_Term + Finals) / 2)|
+-------------------------+
|                    91.25|
|                    82.35|
|        82.19999999999999|
+-------------------------+

sdf4.select((sdf4.Mid_Term/100) * 4).show()

+----------------------+
|((Mid_Term / 100) * 4)|
+----------------------+
|                 3.572|
|                 3.528|
|                 3.052|
+----------------------+
```

Column Alias

```
sdf5 = sdf4.select(sdf4.Subject,((sdf4.Mid_Term +
sdf4.Finals)/2).alias("Semester_Avg"))
sdf5.show()

+---------+-----------------+
|  Subject|     Semester_Avg|
+---------+-----------------+
|  Physics|            91.25|
|Chemistry|            82.35|
|  Biology|82.19999999999999|
+---------+-----------------+
```

Column Sorting

```
sdf6 = sdf5.sort(sdf5.Semester_Avg.asc())
sdf6.show()
```

```
+---------+-----------------+
|  Subject|     Semester_Avg|
+---------+-----------------+
|  Biology|82.19999999999999|
|Chemistry|            82.35|
|  Physics|            91.25|
+---------+-----------------+

sdf7 = sdf5.sort(sdf5.Semester_Avg.desc())
sdf7.show()

+---------+-----------------+
|  Subject|     Semester_Avg|
+---------+-----------------+
|  Physics|            91.25|
|Chemistry|            82.35|
|  Biology|82.19999999999999|
+---------+-----------------+
```

Column Cast

```
sdf8 = sdf5.select(sdf5.Subject,sdf5.Semester_Avg.cast("int"))
sdf8.show()
sdf5.printSchema()
sdf8.printSchema()

+---------+------------+
|  Subject|Semester_Avg|
+---------+------------+
|  Physics|          91|
|Chemistry|          82|
|  Biology|          82|
+---------+------------+

root
 |-- Subject: string (nullable = true)
 |-- Semester_Avg: double (nullable = true)

After Casting the double to int
root
 |-- Subject: string (nullable = true)
 |-- Semester_Avg: integer (nullable = true)
```

Column Filter using >

```
sdf8.filter(sdf8.Semester_Avg > 90).show()
```

```
+-------+------------+
|Subject|Semester_Avg|
+-------+------------+
|Physics|          91|
+-------+------------+
```

Column Filter using between

```
sdf8.filter(sdf8.Semester_Avg.between(80,90)).show()
```

```
+---------+------------+
|  Subject|Semester_Avg|
+---------+------------+
|Chemistry|          82|
|  Biology|          82|
+---------+------------+
```

Column Filter using contains

```
sdf8.filter(sdf8.Subject.contains("Bio")).show()
```

```
+-------+------------+
|Subject|Semester_Avg|
+-------+------------+
|Biology|          82|
+-------+------------+
```

Column Filter using startswith and endswith

```
sdf8.filter(sdf8.Subject.startswith("B")).show()
sdf8.filter(sdf8.Subject.endswith("y")).show()
```

Starts with B

```
+-------+------------+
|Subject|Semester_Avg|
+-------+------------+
|Biology|          82|
+-------+------------+
```

Ends with y

```
+---------+------------+
|  Subject|Semester_Avg|
+---------+------------+
|Chemistry|          82|
|  Biology|          82|
+---------+------------+
```

Column when and otherwise

```
from pyspark.sql.functions import when
sdf9 = sdf7.select(sdf7.Subject,sdf7.Semester_Avg,
                   when(sdf7.Semester_Avg > 90, "A")
                   .when(sdf7.Semester_Avg > 85, "B+")
                   .when(sdf7.Semester_Avg > 80, "B")
                   .otherwise(sdf7.Semester_Avg).alias("Grade"))
sdf9.show()
```

```
+---------+-----------------+-----+
|  Subject|     Semester_Avg|Grade|
+---------+-----------------+-----+
|  Physics|            91.25|    A|
|Chemistry|            82.35|    B|
|  Biology|82.19999999999999|    B|
+---------+-----------------+-----+
```

Ceil and Floor on Column Values

```
from pyspark.sql.functions import ceil,floor
print("Original DataFrame")
sdf7.show()
print("Using Ceil")
sdf7.select(sdf7.Subject,ceil(sdf7.Semester_Avg)).show()
print("Using Floor")
sdf7.select(sdf7.Subject,floor(sdf7.Semester_Avg)).show()
```

Original DataFrame

```
+---------+-----------------+
|  Subject|     Semester_Avg|
+---------+-----------------+
|  Physics|            91.25|
|Chemistry|            82.35|
|  Biology|82.19999999999999|
+---------+-----------------+
```

Using Ceil

```
+---------+------------------+
|  Subject|CEIL(Semester_Avg)|
+---------+------------------+
|  Physics|                92|
|Chemistry|                83|
|  Biology|                83|
+---------+------------------+
```

Using Floor

```
+---------+-------------------+
|  Subject|FLOOR(Semester_Avg)|
+---------+-------------------+
|  Physics|                 91|
|Chemistry|                 82|
|  Biology|                 82|
+---------+-------------------+
```

Round to 2 Decimals

```
from pyspark.sql.functions import round
sdf7.select(sdf7.Subject,round(sdf7.Semester_Avg,2)).show()

+---------+----------------------+
|  Subject|round(Semester_Avg, 2)|
+---------+----------------------+
|  Physics|                 91.25|
|Chemistry|                 82.35|
|  Biology|                  82.2|
+---------+----------------------+
```

Columns Concatenation with and without separator

```
from pyspark.sql.functions import concat,concat_ws,col

eData=[("Debasis","Das",39,1000),
      ("John","Doe",40,2000),
      ("Jane","Doe",41,3000)]
edf =spark.createDataFrame(eData).toDF("First_Name","Last_Name","Age","Salary")
edf.show()

edf1 =
edf.select(concat(edf.First_Name,edf.Last_Name).alias("Full_Name"),edf.Age,edf.Salary
)
edf1.show()

edf2 = edf.select(concat_ws("
",edf.First_Name,edf.Last_Name).alias("Full_Name"),edf.Age,edf.Salary)
edf2.show()
```

```
+----------+---------+---+------+
|First_Name|Last_Name|Age|Salary|
+----------+---------+---+------+
|   Debasis|      Das| 39|  1000|
|      John|      Doe| 40|  2000|
|      Jane|      Doe| 41|  3000|
+----------+---------+---+------+
```

Concat without Separator

```
+----------+---+------+
| Full_Name|Age|Salary|
+----------+---+------+
|DebasisDas| 39|  1000|
|   JohnDoe| 40|  2000|
|   JaneDoe| 41|  3000|
+----------+---+------+
```

Concat with Separator

```
+-----------+---+------+
|  Full_Name|Age|Salary|
+-----------+---+------+
|Debasis Das| 39|  1000|
|   John Doe| 40|  2000|
|   Jane Doe| 41|  3000|
+-----------+---+------+
```

Other Column Operations

```
from pyspark.sql.functions import stddev
data1 = [(100,199),
         (200,299),
         (300,399),
         (400,499),
         (500,599)
        ]
ddf =spark.createDataFrame(data1).toDF("col1","col2")
ddf.show()
ddf.select(ddf.col1,
           ddf.col2,
           (ddf.col1+ddf.col2).alias("add"),
           (ddf.col1-ddf.col2).alias("sub"),
           (ddf.col1*ddf.col2).alias("mul"),
           (ddf.col1/ddf.col2).alias("div"),
           round(ddf.col1/ddf.col2,2).alias("div_roun"),
          ).show()
```

```
+----+----+
|col1|col2|
+----+----+
| 100| 199|
| 200| 299|
| 300| 399|
| 400| 499|
| 500| 599|
+----+----+

+----+----+----+---+------+------------------+--------+
|col1|col2| add|sub|   mul|               div|div_roun|
+----+----+----+---+------+------------------+--------+
| 100| 199| 299|-99| 19900|0.5025125628140703|     0.5|
| 200| 299| 499|-99| 59800|0.6688963210702341|    0.67|
| 300| 399| 699|-99|119700|0.7518796992481203|    0.75|
| 400| 499| 899|-99|199600|0.8016032064128257|     0.8|
| 500| 599|1099|-99|299500|0.8347245409015025|    0.83|
+----+----+----+---+------+------------------+--------+
```

Column Standard Deviation

```
from pyspark.sql.functions import stddev,avg,min,max
ddf.select(avg(ddf.col1),
           stddev(ddf.col1),
           min(ddf.col1),
           max(ddf.col1)).show()

+---------+-----------------+---------+---------+
|avg(col1)| stddev_samp(col1)|min(col1)|max(col1)|
+---------+-----------------+---------+---------+
|    300.0|158.11388300841898|      100|      500|
+---------+-----------------+---------+---------+
```

Column Aggregate – Standard Deviation

```
ddf.agg({'col1': 'stddev', 'col2': 'Stddev'}).show()

+-----------------+-----------------+
|     stddev(col2)|     stddev(col1)|
+-----------------+-----------------+
|274.0439948459209|273.8612787525831|
+-----------------+-----------------+
```

Column Covariance

```
ddf.cov("col1","col2")
#75049.99999999999
```

Additional Reading & References

- https://spark.apache.org/docs/3.1.1/api/java/org/apache/spark/sql/Column.html
- https://ddas.tech/pyspark-groupby-examples/