**PySpark UDF (User Defined Function)**

PySpark UDF Example

PySpark UDF (a.k.a User Defined Function) is the most useful feature of Spark SQL & DataFrame that is used to extend the PySpark build in capabilities. In this article, I will explain what is UDF? why do we need it and how to create and use it on DataFrame select(), withColumn() and SQL using PySpark (Spark with Python) examples.

**Note:** UDF’s are the most expensive operations hence use them only you have no choice and when essential. In the later section of the article, I will explain why using UDF’s is an expensive operation in detail.

**1. PySpark UDF Introduction**

**1.1 What is UDF?**

UDF’s a.k.a User Defined Functions, If you are coming from SQL background, UDF’s are nothing new to you as most of the traditional RDBMS databases support User Defined Functions, these functions need to register in the database library and use them on SQL as regular functions.

PySpark UDF’s are similar to UDF on traditional databases. In PySpark, you create a function in a Python syntax and wrap it with PySpark SQL udf() or register it as udf and use it on DataFrame and SQL respectively.

**1.2 Why do we need a UDF?**

UDF’s are used to extend the functions of the framework and re-use these functions on multiple DataFrame’s. For example, you wanted to convert every first letter of a word in a name string to a capital case; PySpark build-in features don’t have this function hence you can create it a UDF and reuse this as needed on many Data Frames. UDF’s are once created they can be re-used on several DataFrame’s and SQL expressions.

Before you create any UDF, do your research to check if the similar function you wanted is already available in [Spark SQL Functions](https://sparkbyexamples.com/spark/spark-sql-functions-understanding/). PySpark SQL provides several predefined common functions and many more new functions are added with every release. hence, It is best to check before you reinventing the wheel.

When you creating UDF’s you need to design them very carefully otherwise you will come across optimization & performance issues.

**2. Create PySpark UDF**

**2.1 Create a DataFrame**

Before we jump in creating a UDF, first let’s create a PySpark DataFrame.

columns = ["Seqno","Name"]

data = [("1", "john jones"),

("2", "tracey smith"),

("3", "amy sanders")]

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

df.show(truncate=False)

Yields below output.

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

|Seqno|Names |

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

|1 |john jones |

|2 |tracey smith|

|3 |amy sanders |

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

**2.2 Create a Python Function**

The first step in creating a UDF is creating a Python function. Below snippet creates a function convertCase() which takes a string parameter and converts the first letter of every word to capital letter. UDF’s take parameters of your choice and returns a value.

def convertCase(str):

resStr=""

arr = str.split(" ")

for x in arr:

resStr= resStr + x[0:1].upper() + x[1:len(x)] + " "

return resStr

**2.3 Convert a Python function to PySpark UDF**

Now convert this function convertCase() to UDF by passing the function to PySpark SQL udf(), this function is available at org.apache.spark.sql.functions.udf package. Make sure you import this package before using it.

PySpark SQL udf() function returns org.apache.spark.sql.expressions.UserDefinedFunction class object.

""" Converting function to UDF """

convertUDF = udf(lambda z: convertCase(z),StringType())

**Note:** The default type of the udf() is StringType hence, you can also write the above statement without return type.

""" Converting function to UDF

StringType() is by default hence not required """

convertUDF = udf(lambda z: convertCase(z))

**3. Using UDF with DataFrame**

**3.1 Using UDF with PySpark DataFrame select()**

Now you can use convertUDF() on a DataFrame column as a regular build-in function.

df.select(col("Seqno"), \

convertUDF(col("Name")).alias("Name") ) \

.show(truncate=False)

This results below output.

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

|Seqno|Name |

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

|1 |John Jones |

|2 |Tracey Smith |

|3 |Amy Sanders |

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

**3.2 Using UDF with PySpark DataFrame withColumn()**

You could also use udf on DataFrame withColumn() function, to explain this I will create another upperCase() function which converts the input string to upper case.

def upperCase(str):

return str.upper()

Let’s convert upperCase() python function to UDF and then use it with DataFrame withColumn(). Below example converts the values of “Name” column to upper case and creates a new column “Curated Name”

upperCaseUDF = udf(lambda z:upperCase(z),StringType())

df.withColumn("Cureated Name", upperCaseUDF(col("Name"))) \

.show(truncate=False)

This yields below output.

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

|Seqno|Name |Cureated Name|

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

|1 |john jones |JOHN JONES |

|2 |tracey smith|TRACEY SMITH |

|3 |amy sanders |AMY SANDERS |

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

**3.3 Registering PySpark UDF & use it on SQL**

In order to use convertCase() function on PySpark SQL, you need to register the function with PySpark by using spark.udf.register().

""" Using UDF on SQL """

spark.udf.register("convertUDF", convertCase,StringType())

df.createOrReplaceTempView("NAME\_TABLE")

spark.sql("select Seqno, convertUDF(Name) as Name from NAME\_TABLE") \

.show(truncate=False)

This yields the same output as 3.1 example.

**4. Creating UDF using annotation**

In the previous sections, you have learned creating a UDF is a 2 step process, first, you need to create a Python function, second convert function to UDF using SQL udf() function, however, you can avoid these two steps and create it with just a single step by using annotations.

@udf(returnType=StringType())

def upperCase(str):

return str.upper()

df.withColumn("Cureated Name", upperCase(col("Name"))) \

.show(truncate=False)

This results same output as section 3.2

**5. Special Handling**

**5.1 Execution order**

One thing to aware is in PySpark/Spark does not guarantee the order of evaluation of subexpressions meaning expressions are not guarantee to evaluated left-to-right or in any other fixed order. PySpark reorders the execution for query optimization and planning hence, AND, OR, WHERE and HAVING expression will have side effects.

So when you are designing and using UDF, you have to be very careful especially with null handling as these results runtime exceptions.

"""

No guarantee Name is not null will execute first

If convertUDF(Name) like '%John%' execute first then

you will get runtime error

"""

spark.sql("select Seqno, convertUDF(Name) as Name from NAME\_TABLE " + \

"where Name is not null and convertUDF(Name) like '%John%'") \

.show(truncate=False)

**5.2 Handling null check**

UDF’s are error-prone when not designed carefully. for example, when you have a column that contains the value null on some records

""" null check """

columns = ["Seqno","Name"]

data = [("1", "john jones"),

("2", "tracey smith"),

("3", "amy sanders"),

('4',None)]

df2 = spark.createDataFrame(data=data,schema=columns)

df2.show(truncate=False)

df2.createOrReplaceTempView("NAME\_TABLE2")

spark.sql("select convertUDF(Name) from NAME\_TABLE2") \

.show(truncate=False)

Note that from the above snippet, record with “Seqno 4” has value “None” for “name” column. Since we are not handling null with UDF function, using this on DataFrame returns below error. Note that in Python None is considered null.

AttributeError: 'NoneType' object has no attribute 'split'

at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.handlePythonException(PythonRunner.scala:456)

at org.apache.spark.sql.execution.python.PythonUDFRunner$$anon$1.read(PythonUDFRunner.scala:81)

at org.apache.spark.sql.execution.python.PythonUDFRunner$$anon$1.read(PythonUDFRunner.scala:64)

at org.apache.spark.api.python.BasePythonRunner$ReaderIterator.hasNext(PythonRunner.scala:410)

at org.apache.spark.InterruptibleIterator.hasNext(InterruptibleIterator.scala:37)

at scala.collection.Iterator$$anon$12.hasNext(Iterator.scala:440)

Below points to remember

* Its always best practice to check for null inside a UDF function rather than checking for null outside.
* In any case, if you can’t do a null check in UDF at lease use IF or CASE WHEN to check for null and call UDF conditionally.

spark.udf.register("\_nullsafeUDF", lambda str: convertCase(str) if not str is None else "" , StringType())

spark.sql("select \_nullsafeUDF(Name) from NAME\_TABLE2") \

.show(truncate=False)

spark.sql("select Seqno, \_nullsafeUDF(Name) as Name from NAME\_TABLE2 " + \

" where Name is not null and \_nullsafeUDF(Name) like '%John%'") \

.show(truncate=False)

This executes successfully without errors as we are checking for null/none while registering UDF.

**5.3 Performance concern using UDF**

UDF’s are a black box to PySpark hence it can’t apply optimization and you will lose all the optimization PySpark does on Dataframe/Dataset. When possible you should use Spark SQL built-in functions as these functions provide optimization. Consider creating UDF only when existing built-in SQL function doesn’t have it.

**6. Complete PySpark UDF Example**

Below is complete UDF function example in Scala

import pyspark

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, udf

from pyspark.sql.types import StringType

columns = ["Seqno","Name"]

data = [("1", "john jones"),

("2", "tracey smith"),

("3", "amy sanders")]

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

df.show(truncate=False)

def convertCase(str):

resStr=""

arr = str.split(" ")

for x in arr:

resStr= resStr + x[0:1].upper() + x[1:len(x)] + " "

return resStr

""" Converting function to UDF """

convertUDF = udf(lambda z: convertCase(z))

df.select(col("Seqno"), \

convertUDF(col("Name")).alias("Name") ) \

.show(truncate=False)

def upperCase(str):

return str.upper()

upperCaseUDF = udf(lambda z:upperCase(z),StringType())

df.withColumn("Cureated Name", upperCaseUDF(col("Name"))) \

.show(truncate=False)

""" Using UDF on SQL """

spark.udf.register("convertUDF", convertCase,StringType())

df.createOrReplaceTempView("NAME\_TABLE")

spark.sql("select Seqno, convertUDF(Name) as Name from NAME\_TABLE") \

.show(truncate=False)

spark.sql("select Seqno, convertUDF(Name) as Name from NAME\_TABLE " + \

"where Name is not null and convertUDF(Name) like '%John%'") \

.show(truncate=False)

""" null check """

columns = ["Seqno","Name"]

data = [("1", "john jones"),

("2", "tracey smith"),

("3", "amy sanders"),

('4',None)]

df2 = spark.createDataFrame(data=data,schema=columns)

df2.show(truncate=False)

df2.createOrReplaceTempView("NAME\_TABLE2")

spark.udf.register("\_nullsafeUDF", lambda str: convertCase(str) if not str is None else "" , StringType())

spark.sql("select \_nullsafeUDF(Name) from NAME\_TABLE2") \

.show(truncate=False)

spark.sql("select Seqno, \_nullsafeUDF(Name) as Name from NAME\_TABLE2 " + \

" where Name is not null and \_nullsafeUDF(Name) like '%John%'") \

.show(truncate=False)

T