Introduction

The Python API for Apache Spark is known as PySpark.To develop spark applications in Python, we will use PySpark. It also provides the Pyspark shell for real-time data analysis. PySpark supports most of the Apache Spark functionality, including Spark Core, SparkSQL, DataFrame, Streaming, MLlib (Machine Learning), and [MLlib](https://www.analyticsvidhya.com/blog/2022/08/complete-guide-to-run-machine-learning-on-spark-using-spark-mllib/" \t "_blank) (Machine Learning).

This article will explore useful PySpark functions with scenario-based examples to understand them better.

The expr() function

It is a SQL function in PySpark to 𝐞𝐱𝐞𝐜𝐮𝐭𝐞 𝐒𝐐𝐋-𝐥𝐢𝐤𝐞 𝐞𝐱𝐩𝐫𝐞𝐬𝐬𝐢𝐨𝐧𝐬. It will accept a SQL expression as a string argument and execute the commands written in the statement. It enables the use of SQL-like functions that are absent from the PySpark Column type and pyspark.sql.functions API. Ex:- 𝐂𝐀𝐒𝐄 𝐖𝐇𝐄𝐍. We are allowed to use 𝐃𝐚𝐭𝐚𝐅𝐫𝐚𝐦𝐞 𝐜𝐨𝐥𝐮𝐦𝐧𝐬 in the expression. The syntax for this function is 𝐞𝐱𝐩(𝐬𝐭𝐫).

# importing necessary libs

from pyspark.sql import SparkSession

from pyspark.sql.functions import expr

# creating session

spark = SparkSession.builder.appName("practice").getOrCreate()

# create data

data = [("Prashant","Banglore",25, 58, "2022-08-01", 1),

("Ankit","Banglore",26,54,"2021-05-02", 2),

("Ramakant","Gurugram",24, 60, "2022-06-02", 3),

("Brijesh","Gazipur", 26,75,"2022-07-04", 4),

("Devendra","Gurugram", 27, 62, "2022-04-03", 5),

("Ajay","Chandigarh", 25,72,"2022-02-01", 6)]

columns= ["friends\_name","location", "age", "weight", "meetup\_date", "offset"]

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

df\_friends.show()



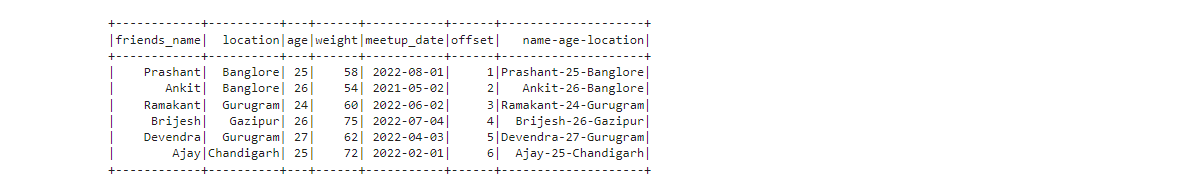
**Let’s see the practical Implementations:-**

**Example:- A.) Concatenating one or more columns using expr()**

# concate friend's name, age, and location columns using expr()

df\_concat = df\_friends.withColumn("name-age-location",expr("friends\_name|| '-'|| age || '-' || location"))

df\_concat.show()



We have joined the name, age, and location columns and stored the result in a new column called “name-age-location.”

**Example:- B.) Add a new column based on a condition (CASE WHEN) using expr()**

# check if exercise needed based on weight

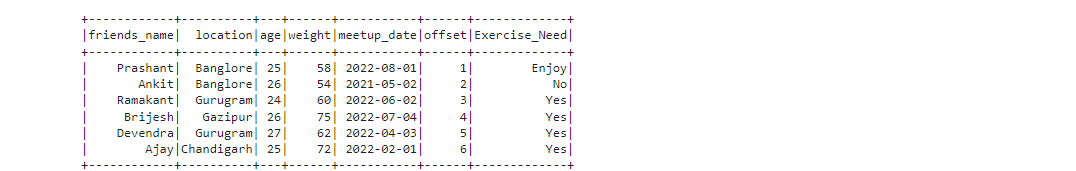
# if weight is more or equal to 60 -- Yes

# if weight is less than 55 -- No

# else -- Enjoy

df\_condition = df\_friends.withColumn("Exercise\_Need", expr("CASE WHEN weight >= 60 THEN 'Yes' " + "WHEN weight < 55 THEN 'No' ELSE 'Enjoy' END"))

df\_condition.show()



Our **“Exercise\_Need”** column received three values **(Enjoy, No, and Yes)** based on the condition given in **CASE WHEN**. The first value of the weight column is 58, so it’s less than 60 and more than 55, so the result is **“Enjoy.”**

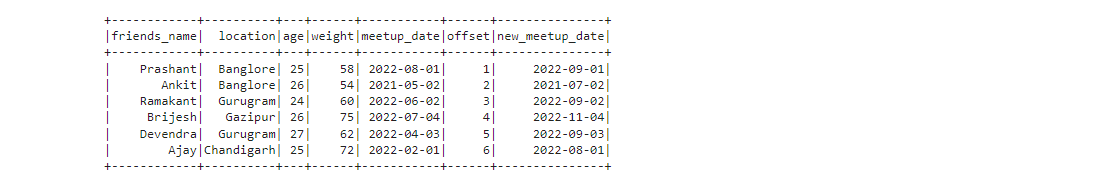
**Example:- C.) Creating a new column using the current column value inside the expression.**



**Ready to start your data science journey?**

**Master 23+ tools & learn with guided projects with the Certified AI & ML BlackBelt Plus Program**

[Enroll Now](https://blackbelt.analyticsvidhya.com/plus?utm_source=blog_india&utm_medium=desktop_banner_between_articles&utm_campaign=01-Oct-2023%7C%7C&utm_content=tools)



The**“meetup\_date”** month value increases by the offset value, and the newly generated result is stored in the**“new\_meetup\_date”**column.

The Padding Functions

**A.) lpad():-**

This function provides padding to the left side of the column, and the inputs for this function are column name, length, and padding string.

**B.) rpad ():-**

This function is used to add padding to the right side of the column. Column name, length, and padding string are additional inputs for this function.

**Note:-**

* If the column value is longer than the specified length, the return value will be shortened to length characters or bytes.
* If the padding value is not specified, then the column value will be padded to the left or right depending on the function you are using, with space characters if it is a character string and with zeros if it is a byte sequence.

**Let’s first create a data Frame:-**

# importing necessary libs

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, lpad, rpad

# creating session

spark = SparkSession.builder.appName("practice").getOrCreate()

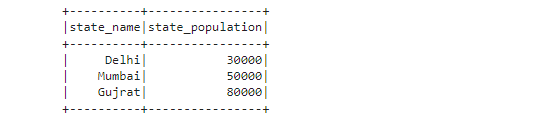
# creating data

data = [("Delhi",30000),("Mumbai",50000),("Gujrat",80000)]

columns= ["state\_name","state\_population"]

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

df\_states.show()



**Example:- 01 – Use of left padding**

# left padding

df\_states = df\_states.withColumn('states\_name\_leftpad', lpad(col("state\_name"), 10, '#'))

df\_states.show(truncate =False)



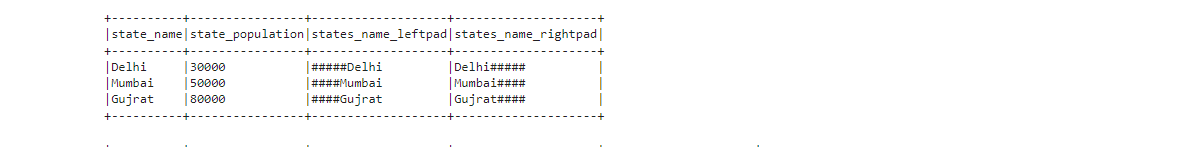
We added the **‘#’**symbol to the left of the “**state\_name”** column values, and the total length of column values becomes “**10″** after the padding.

**Example:-02 – Right padding**

# right padding

df\_states = df\_states.withColumn('states\_name\_rightpad', rpad(col("state\_name"), 10, '#'))

df\_states.show(truncate =False)

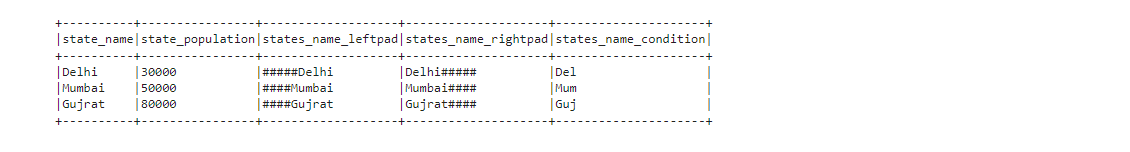


We added the **“#”** symbol to the right of the “state\_name” column values, and the total length becomes ten after the right padding.

**Example:-03 – When the column string length is longer than the padded string length**

df\_states = df\_states.withColumn('states\_name\_condition', lpad(col("state\_name"), 3, '#'))

df\_states.show(truncate =False)



In this case, the return column value will be shortened to the length of the padded string length. You can see the **“state\_name\_condition”** column only has values of length 3, which is the **padded length** we have given in the function.

The repeat() Function

In PySpark, we use the repeat function to duplicate the column values. The repeat(str,n) function returns the string containing the specified string value repeated n times.

**Example:- 01**

# importing necessary libs

from pyspark.sql import SparkSession

from pyspark.sql.functions import expr, repeat

# creating session

spark = SparkSession.builder.appName("practice").getOrCreate()

# # create data

data = [("Prashant",25, 80), ("Ankit",26, 90),("Ramakant", 24, 85)]

columns= ["student\_name", "student\_age", "student\_score"]

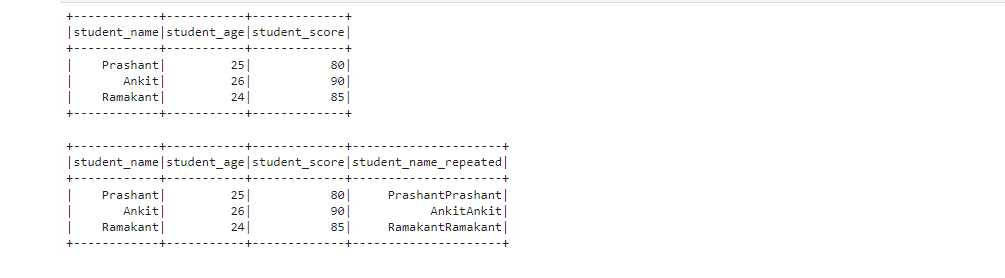
df\_students = spark.createDataFrame(data = data, schema = columns)

df\_students.show()

# repeating the column (student\_name) twice and saving results in new column

df\_repeated = df\_students.withColumn("student\_name\_repeated",(expr("repeat(student\_name, 2)")))

df\_repeated.show()



We have repeated the “student\_name” column values in the above example **twice**.

We can also use this function with the Concat function, where we can repeat some string values n times before column values, working like padding, where n may be the length of some values.

The startswith() and endswith() function

**startswith():-**

It will produce a boolean result of True or False. When the Dataframe column value ends with the string provided as a parameter to this method, it returns True. If no match is found, it returns False.

**endswith():-**

The boolean value (True/False) will be returned. When the DataFrame column value ends with a string supplied as an input to this method, it returns True. False is returned if not matched.

**Note:-**

* Return 𝐍𝐔𝐋𝐋 if either of the column values or input strings are 𝐍𝐔𝐋𝐋.
* Return 𝗧𝗿𝘂𝗲 if the input check strings are empty.
* These methods are case-sensitive.

**Create a data frame:-**

# importing necessary libs

from pyspark.sql import SparkSession

from pyspark.sql.functions import col

# creating session

spark = SparkSession.builder.appName("practice").getOrCreate()

# # create dataframe

data = [("Prashant",25, 80), ("Ankit",26, 90),("Ramakant", 24, 85), (None, 23, 87)]

columns= ["student\_name", "student\_age", "student\_score"]

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

df\_students.show()



**Example – 01  First, check the output type.**

df\_internal\_res = df\_students.select(col("student\_name").endswith("it").alias("internal\_bool\_val"))

df\_internal\_res.show()



* The output is a boolean value.
* The output value is null for the last row value because the corresponding value of the “students\_name” column is NULL.

**Example – 02**

* Now we use the filter() method to fetch the rows corresponding to the True values.

df\_check\_start = df\_students.filter(col("student\_name").startswith("Pra"))

df\_check\_start.show()



Here we got the first row as output because the “student\_name” column value starts with the value mentioned inside the function.

**Example – 03**

df\_check\_end = df\_students.filter(col("student\_name").endswith("ant"))

df\_check\_end.show()



Here we got the two rows as output because the “student\_name” column value ends with the value mentioned inside the function.

**Example – 04 – What if arguments in functions are empty?**

df\_check\_empty = df\_students.filter(col("student\_name").endswith(""))

df\_check\_empty.show()



In this case, we get a True value corresponding to each row, and no False value returned