Learning Substring Extraction in PySpark: A Comprehensive Guide

Authored by

Mohammed looti

November 11, 2025

String manipulation is a fundamental requirement in data engineering and analysis. When working with large datasets using PySpark, extracting specific portions of text--or substrings--from a column in a DataFrame is a common task. PySpark provides powerful, optimized functions within the `pyspark.sql.functions` module to handle these operations efficiently.

We will explore five essential techniques for substring extraction, primarily utilizing the `F.substring` and `F.substring_index` functions. These methods allow you to precisely target the required segment of a string based on position or delimiter.

Method 1: Extract Substring from the Beginning of the String

To extract characters starting from the very first position, we utilize the `F.substring(col, pos, len)` function. When extracting from the beginning, the starting position (`pos`) is always 1.

from pyspark.sql import functions as F

#extract first three characters from team column
df_new = df.withColumn('first3', F.substring('team', 1, 3))

Method 2: Extract Substring from the Middle of the String

The `F.substring` function is highly flexible and allows extraction from any arbitrary point within the string. We simply adjust the `pos` parameter to indicate where the extraction should begin and define the desired length of the segment.

from pyspark.sql import functions as F

#extract four characters starting from position two in team column
df_new = df.withColumn('mid4', F.substring('team', 2, 4))

Method 3: Extract Substring from the End of the String

While `F.substring` typically uses positive indices, it also supports negative indexing, similar to standard Python string operations. A negative starting position indicates counting backwards from the end of the string.

from pyspark.sql import functions as F

#extract last three characters from team column
df_new = df.withColumn('last3', F.substring('team', -3, 3))

Method 4: Extract Substring Before a Specific Delimiter

When extraction needs to be based on the presence of a specific character (a delimiter) rather than a fixed position, we use the highly efficient `F.substring_index(str, delim, count)` function. To extract the content before the first occurrence of a delimiter, we set the `count` parameter to `1`.

from pyspark.sql import functions as F

#extract all characters before space in team column
df_new = df.withColumn('beforespace', F.substring_index('team', ' ', 1))

Method 5: Extract Substring After a Specific Delimiter

If the goal is to retrieve the text that appears after a delimiter, we again leverage `F.substring_index`, but this time we use a negative value for the `count` parameter. Setting `count` to `-1` returns the substring following the last occurrence of the specified delimiter.

from pyspark.sql import functions as F

#extract all characters after space in team column
df_new = df.withColumn('afterspace', F.substring_index('team', ' ', -1))

Setting up the PySpark DataFrame for Examples

To demonstrate these five substring extraction methods in action, we must first initialize a PySpark session and create a sample dataset. We will use a simple list of basketball teams and their points, where the string manipulation will occur on the `team` column.

from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

# Define the input data containing team names and points
data = ,
,
,
,
,
]

```
# Define column names
columns =

# Create the PySpark DataFrame
df = spark.createDataFrame(data, columns)

# Display the initial DataFrame structure
df.show()

+--------------+------+
| team|points|
+--------------+------+
| Dallas Mavs| 18|
| Brooklyn Nets| 33|
| Atlanta Hawks| 12|
|Boston Celtics| 15|
| Miami Heat| 19|
|Cleveland Cavs| 24|
| Orlando Magic| 28|
+--------------+------+
```

Example 1: Isolating Initial Characters (Start of String)

This example demonstrates how to extract the first three characters from the `team` column, effectively creating an abbreviation for each team name. We use `F.substring(column, 1, 3)` to define the starting point (**1**) and the desired length (**3**).

```
from pyspark.sql import functions as F

#extract first three characters from team column
df_new = df.withColumn('first3', F.substring('team', 1, 3))

#view updated DataFrame
df_new.show()

+--------------+------+------+
| team|points|first3|
+--------------+------+------+
| Dallas Mavs| 18| Dal|
| Brooklyn Nets| 33| Bro|
| Atlanta Hawks| 12| Atl|
```

|Boston Celtics| 15| Bos|
| Miami Heat| 19| Mia|
|Cleveland Cavs| 24| Cle|
| Orlando Magic| 28| Orl|
+--------------+------+------+

Example 2: Extracting Characters from the Middle Position

To extract a segment starting internally, we adjust the position index. In this case, we start at position **2** and pull four characters. Remember that PySpark string indexing is 1-based, meaning position 2 is the second character of the string.

from pyspark.sql import functions as F

#extract four characters starting from position two in team column
df_new = df.withColumn('mid4', F.substring('team', 2, 4))

#view updated DataFrame
df_new.show()

+--------------+------+----+
| team|points|mid4|
+--------------+------+----+
Dallas Mavs	18	alla
Brooklyn Nets	33	rook
Atlanta Hawks	12	tlan
Boston Celtics	15	osto
Miami Heat	19	iami
Cleveland Cavs	24	leve
Orlando Magic	28	rlan
+--------------+------+----+

Example 3: Capturing Trailing Characters (End of String)

Extracting the end of a string is handled by passing a negative index to the `pos` parameter of the `F.substring` function. We will extract the last three characters from the team name by specifying **-3** as the starting position and a length of **3**.

from pyspark.sql import functions as F

```
#extract last three characters from team column
df_new = df.withColumn('last3', F.substring('team', -3, 3))

#view updated DataFrame
df_new.show()

+--------------+------+-----+
| team|points|last3|
+--------------+------+-----+
| Dallas Mavs| 18| avs|
| Brooklyn Nets| 33| ets|
| Atlanta Hawks| 12| wks|
|Boston Celtics| 15| ics|
| Miami Heat| 19| eat|
|Cleveland Cavs| 24| avs|
| Orlando Magic| 28| gic|
+--------------+------+-----+
```

Example 4: Splitting Strings Before a Delimiter

When cleaning data, it is often necessary to isolate the first word or component of a multi-word string. Using `F.substring_index` with a positive count (**1**) allows us to grab everything before the specified delimiter (in this case, a space `' '`).

from pyspark.sql import functions as F

```
#extract all characters before space in team column
df_new = df.withColumn('beforespace', F.substring_index('team', ' ', 1))

#view updated DataFrame
df_new.show()

+--------------+------+-----------+
| team|points|beforespace|
+--------------+------+-----------+
| Dallas Mavs| 18| Dallas|
| Brooklyn Nets| 33| Brooklyn|
| Atlanta Hawks| 12| Atlanta|
|Boston Celtics| 15| Boston|
| Miami Heat| 19| Miami|
|Cleveland Cavs| 24| Cleveland|
```

| Orlando Magic| 28| Orlando|
+--------------+------+-----------+

Example 5: Splitting Strings After a Delimiter

To extract the subsequent part of the string--the content after the delimiter--we use `F.substring_index` with a negative count (**-1**). This is ideal for isolating the second component of a compound name, such as the mascot or city name from the team column.

from pyspark.sql import functions as F

#extract all characters after space in team column
df_new = df.withColumn('afterspace', F.substring_index('team', ' ', -1))

#view updated DataFrame
df_new.show()

+--------------+------+----------+
| team|points|afterspace|
+--------------+------+----------+
Dallas Mavs	18	Mavs
Brooklyn Nets	33	Nets
Atlanta Hawks	12	Hawks
Boston Celtics	15	Celtics
Miami Heat	19	Heat
Cleveland Cavs	24	Cavs
Orlando Magic	28	Magic
+--------------+------+----------+

Further PySpark String Manipulation Resources

Mastering string functions is essential for effective data cleaning and preparation within the PySpark environment. The techniques demonstrated here using `F.substring` and `F.substring_index` provide robust solutions for both fixed-length and delimiter-based extraction problems.

If you are looking to expand your knowledge of DataFrame transformations, consider exploring the following related tutorials covering other common tasks:

How to handle null values in PySpark columns.

Applying regular expressions for advanced pattern matching in strings.

Efficiently joining multiple PySpark DataFrames.