How to Count Null Values in PySpark DataFrames

Authored by

stats writer

February 8, 2026

ARABPSYCHOLOGY.COM

RECOMMENDED CITATION

stats writer (2026). *How to Count Null Values in PySpark DataFrames*. PSYCHOLOGICAL SCALES. Retrieved from https://scales.arabpsychology.com/?p=129809

In the realm of large-scale data processing, identifying and managing missing data is a critical first step towards reliable analysis. In PySpark, the powerful Python API for Apache Spark, detecting missing entries, often represented as null values (1), requires specific vectorized functions designed for distributed computation. To efficiently calculate the total occurrence of missing data points within a specific column or across the entire dataset contained within a PySpark DataFrame (1), developers typically leverage a combination of built-in functions, primarily the `isNull()` function paired with aggregation functions like `sum()` or conditional filtering.

This approach allows for quick and precise diagnostics of data quality issues across potentially massive datasets, which is essential before performing any advanced machine learning or analytical operations. For instance, if a DataFrame named `df` contains crucial metrics, the syntax `df.select(sum(isNull("column_name"))).show()` provides a direct count of nulls in that designated column. Furthermore, slightly more complex conditional logic or iteration allows us to extend this technique to generate a comprehensive summary of missingness for every single column simultaneously, thereby offering a complete data quality overview necessary for robust data engineering workflows.

Count Null Values in PySpark (With Examples)

Introduction to Null Value Handling in PySpark

Working with real-world data invariably means encountering missing or incomplete records. In the context of big data processing using Spark, these missing entries are typically represented by null values (2) within a PySpark DataFrame (2). Effective data quality management necessitates an accurate and efficient mechanism for quantifying these nulls, enabling data engineers to decide whether to impute, drop, or flag records for further investigation.

PySpark provides robust, optimized methods for this crucial task, leveraging its distributed architecture to ensure scalability even when dealing with terabytes of information. The fundamental techniques rely on transforming the detection of a null value into a binary operation (True/False, or 1/0), which can then be aggregated across the entire partition structure of the DataFrame. Understanding these core functions--specifically the conditional checking capabilities and aggregation logic--is paramount to mastering data cleaning in a Spark environment.

Core PySpark Functions for Null Detection

The primary tool utilized for identifying missing data in PySpark is the `isNull` (1) function. When applied to a column, this function evaluates each row and returns a Boolean result: `True` if the cell contains a null value (3), and `False` otherwise. While this function efficiently identifies the presence of nulls, simply listing the Boolean results is not practical for counting them across millions of rows.

ARABPSYCHOLOGY.COM

To obtain a quantifiable count, the output of `isNull` (2) must be aggregated. PySpark handles Boolean values in arithmetic operations by treating `True` as 1 and `False` as 0. This characteristic makes the `sum` (1) function the ideal companion to `isNull()`. By applying `sum()` to the results of `isNull()`, we effectively count every instance where the condition is met (i.e., where a null value exists), yielding the total number of missing entries for that column.

Prerequisite: Setting Up the Sample PySpark DataFrame

To demonstrate the various methods for counting null values, we first need to establish a working environment and create a sample PySpark DataFrame (3). This example uses information relating to basketball players, including their team, assists, and points, intentionally introducing missing data points (`None` or `null`) to simulate real-world data quality issues. We initiate a SparkSession (1) which serves as the entry point for using Spark functionality.

The following setup script defines the data structure, specifies the column names, and constructs the DataFrame. Viewing the resulting DataFrame confirms the presence of null values in the `assists` and `points` columns, which we will target in our subsequent analyses.

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

#define data
data = ,
,
,
,
,
,
,
,
,
]

#define column names
columns =

#create dataframe using data and column names
df = spark.createDataFrame(data, columns)

#view dataframe
df.show()

ARABPSYCHOLOGY.COM

```
+----+-------+------+
|team|assists|points|
+----+-------+------+
| A| null| 11|
| A| 4| 8|
| A| 2| 22|
| A| 10| null|
| B| 8| null|
| B| 11| 14|
| B| 14| 13|
| B| 6| 7|
| C| 2| 8|
| C| 2| 5|
+----+-------+------+
```

Method 1: Counting Null Values in a Single Column (Filtering Approach)

The first and most straightforward method for counting missing values focuses on a single column using direct filtering. This technique utilizes the DataFrame's `where()` clause combined with the `isNull` (3) function to isolate records containing null values (4) in the column of interest. Once these specific rows are filtered out, the standard DataFrame `count()` action is invoked to return the total number of matching rows.

This approach is highly readable and efficient when the goal is specifically to diagnose the missingness of one particular feature. If we want to determine how many players are missing data in the `points` column, we apply the filter directly to that column. This filtering mechanism is conceptually similar to a SQL `WHERE IS NULL` clause, translating the conditional check into a Spark operation that can be executed across the cluster.

We will apply this method to the `points` column of our sample DataFrame `df` to determine its data completeness status. Note that the output of this command will be a single numeric value representing the total count.

#count number of null values in 'points' column
df.where(df.points.isNull()).count()

Method 1: Practical Example and Visualization

Executing the command above on the sample data yields the following numeric result, confirming

ARABPSYCHOLOGY.COM

the presence of two missing values in the `points` metric. This confirms that two of our ten records are incomplete with respect to the player's scoring data, potentially requiring subsequent data imputation or removal.

#count number of null values in 'points' column
df.where(df.points.isNull()).count()

2

It is often beneficial, particularly during the initial data exploration phase, not just to know the count of the missing records but also to inspect the records themselves. Understanding which rows contain null values (5) can provide context regarding the cause of the missingness (e.g., if nulls are concentrated within a specific team or category). To visualize these specific rows, we simply replace the terminal `count()` action with the `show()` action, which executes the filtering and displays the resulting subset DataFrame.

By replacing `count()` with `show()`, the resulting PySpark DataFrame (4) contains only those records with null values in the `points` column. This visualization confirms that the rows corresponding to Team A and Team B have missing point values.

#display rows with null values in 'points' column
df.where(df.points.isNull()).show()

```
+----+-------+------+
|team|assists|points|
+----+-------+------+
| A| 10| null|
| B| 8| null|
+----+-------+------+
```

Method 2: Counting Null Values Across All Columns (Comprehensive Aggregation)

While Method 1 is excellent for single-column analysis, data quality checks often require a holistic view of missingness across the entire dataset. To achieve this, we employ a more advanced technique that iterates through all columns and uses conditional aggregation to sum up the null occurrences. This method provides a condensed summary table detailing the null count for every feature simultaneously.

This comprehensive method relies on importing specific functions from `pyspark.sql.functions`:

ARABPSYCHOLOGY.COM

`when`, `count`, and `col`. The core logic involves using a Python list comprehension to iterate over `df.columns`. For each column `c`, we construct a conditional expression: `when(col(c).isNull(), c)`. This expression dictates that if the current column value is null (checked by `col(c).isNull()`), we return the column name `c`; otherwise, it is ignored.

Crucially, the `count` (1) function is applied to this conditional output. In PySpark's SQL functions, `count()` only tallies non-null expressions. By structuring the `when` clause such that only null inputs result in a non-null output (the column name `c`), we effectively invert the process, causing `count()` to count only the null entries. Finally, `.alias(c)` ensures the output column retains the original column name, resulting in a clear summary table.

from pyspark.sql.functions import when, count, col

#count number of null values in each column of DataFrame
df.select().show()

Method 2: Practical Implementation and Results

Upon execution of the comprehensive aggregation script, the combination of `when` and `count` returns a single-row DataFrame summarizing the total null counts for every column present in the original dataset `df`. This summary immediately flags which variables require immediate attention for data cleaning or imputation strategies.

from pyspark.sql.functions import when, count, col

#count number of null values in each column of DataFrame
df.select().show()

```
+----+-------+------+
|team|assists|points|
+----+-------+------+
| 0| 1| 2|
+----+-------+------+
```

The resulting output clearly indicates the distribution of missing values across the dataset. Analyzing this comprehensive result, we can conclude:

There are **0** null values in the **team** column, indicating it is completely populated.
There is **1** null value in the **assists** column.
There are **2** null values in the **points** column.

ARABPSYCHOLOGY.COM

This method provides the most effective snapshot of data quality across large datasets, allowing engineers to quickly prioritize remediation efforts based on the severity of missingness in different features.

Conclusion and Next Steps in Data Cleaning

Identifying and quantifying nulls within a PySpark DataFrame (5) is a fundamental skill in big data analytics, serving as the gateway to effective data cleaning and preparation. Whether utilizing the targeted filtering approach (Method 1) or the comprehensive aggregation method (Method 2), PySpark offers optimized functions like `isNull` (4), `sum` (2), and `when` clauses to handle this task at scale.

Once null values are counted, the subsequent steps in the data pipeline typically involve managing these missing entries. Strategies may include dropping rows where critical data is absent, imputing numerical columns using the mean or median calculated across the column using specialized PySpark functions, or transforming the data type if the nulls represent non-standard missing indicators. Mastering these counting techniques is the necessary foundation for building robust and reliable distributed data processing applications.

For those looking to expand their PySpark knowledge further, exploring advanced topics such as filling nulls with specific values (using `fillna()`) or integrating these checks into automated data quality validation pipelines is highly recommended. The ability to efficiently diagnose data completeness ensures that all subsequent analytical models operate on the highest quality data possible.

The following tutorials explain how to perform other common tasks in PySpark:

ARABPSYCHOLOGY.COM