How to Fill Null Values in PySpark DataFrames with Column Medians

Authored by

stats writer

February 5, 2026

PySpark: Filling Null Values with Column Median for Robust Data Imputation

PySpark serves as an indispensable framework for handling vast quantities of data within a distributed computing environment. When dealing with real-world datasets, encountering missing or null values is a common challenge that must be addressed before analysis can commence. Proper handling of these missing data points, often through a technique known as imputation, is critical for maintaining data integrity and ensuring the accuracy of subsequent modeling efforts. While simple methods like replacing nulls with the mean exist, using a statistically robust measure like the column median is often preferred, particularly when dealing with data that may contain outliers.

This article provides a comprehensive guide on how to efficiently calculate the median for specified columns in a PySpark DataFrame and subsequently use those median values to replace all corresponding null entries. PySpark facilitates this complex task through powerful, built-in functions, offering a scalable and reliable solution for large-scale data cleaning and preparation. By mastering this imputation technique, users can ensure their datasets are complete, reliable, and optimally structured for deep analytical processes.

Understanding the Necessity of Median Imputation

When preparing data for machine learning models or statistical analysis, missing data can halt processing or, worse, introduce bias if not handled correctly. Although replacing nulls with the mean is a straightforward approach, the mean is highly sensitive to extreme values or outliers. If a column contains several unusually large or small values, the calculated mean may not accurately represent the typical value of the dataset, leading to inaccurate imputation.

The median, conversely, represents the 50th percentile of the data--the value separating the higher half from the lower half. Because the median is based on the rank order of values rather than their magnitude, it is inherently robust to outliers. This characteristic makes median imputation a superior choice when dealing with skewed distributions or data known to contain measurement errors, ensuring that the imputed values are statistically sound representations of the central tendency of the respective features.

The PySpark Approach to Calculating Medians

PySpark's distributed nature requires a specific approach to calculate statistics like the median across potentially millions or billions of rows. We cannot simply iterate through the data. Instead, we leverage the optimization capabilities of the Spark engine. The process involves two primary

steps: first, calculating the median for the required columns using the aggregation functions; and second, using the resulting dictionary of medians to fill the null values in the original DataFrame.

The specific syntax provided below defines a reusable function that abstracts this multi-step calculation into a clean, single operation. This function utilizes the `median` function imported from `pyspark.sql.functions`, which efficiently computes the median across the distributed partitions of the DataFrame. This method is highly scalable and ensures performance even with massive data volumes typical in PySpark environments.

Implementing the Median Imputation Function

To effectively fill null values with the column median in a PySpark DataFrame, we first need to define a function that handles the aggregation and subsequent filling steps. This function streamlines the process and allows for easy application across multiple datasets. The `df.agg()` method is key here, as it executes the `median()` calculation across the specified columns, returning a new DataFrame containing only the calculated medians.

from pyspark.sql.functions import median

#define function to fill null values with column median
def fillna_median(df, include=set()):
medians = df.agg(*(
median(x).alias(x) for x in df.columns if x in include
))
return df.fillna(medians.first().asDict())

#fill null values with median in specific columns
df = fillna_median(df,)

This powerful code snippet encapsulates the entire imputation logic. The `medians` variable captures the aggregated results, which is a single-row DataFrame where columns are named after the original features, holding their respective median values. We then use `medians.first().asDict()` to convert this result into a Python dictionary, which is the required input format for PySpark's built-in `fillna()` method. The `fillna()` method then automatically maps these calculated values to the corresponding columns in the original DataFrame where null values exist. In this particular example, we are targeting the `points` and `assists` columns, replacing their null entries with their respective column medians.

Example: How to Fill Null Values with Median in PySpark

Practical PySpark Example Setup: Creating the Dataset

To demonstrate the functionality of the `fillna_median` function, let us construct a sample PySpark DataFrame. This dataset represents basketball player statistics and intentionally includes missing data points to simulate a real-world scenario where null values need to be handled. We use `SparkSession` to initialize our Spark environment and define both the data rows and the column schema.

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()

+----+----------+------+-------+
|team|conference|points|assists|
+----+----------+------+-------+
A	East	11	4
A	East	8	9
A	East	10	3
B	West	6	12
B	West	null	2
C	East	5	null
+----+----------+------+-------+

Upon reviewing the output of `df.show()`, we can clearly observe that both the `points` and `assists` columns contain one missing entry each. Specifically, the fifth record for Team B has a null value in the `points` column, and the final record for Team C has a null value in the `assists` column. Our next step is to apply the defined `fillna_median` function to these columns to perform robust imputation.

Executing the Imputation Process

Now that the data is prepared, we execute the previously defined logic using `pyspark.sql.functions.median`. The distributed processing power of PySpark ensures that even if this dataset were millions of rows large, the calculation of the column medians would be highly optimized and fast. This operation effectively cleans the data by replacing the non-existent values with statistically derived estimates.

from pyspark.sql.functions import median

#define function to fill null values with column median
def fillna_median(df, include=set()):
medians = df.agg(*(
median(x).alias(x) for x in df.columns if x in include
))
return df.fillna(medians.first().asDict())

#fill null values with median in specific columns
df = fillna_median(df,)

#view updated DataFrame
df.show()

+----+----------+------+-------+
|team|conference|points|assists|
+----+----------+------+-------+
A	East	11	4
A	East	8	9
A	East	10	3
B	West	6	12
B	West	8	2
C	East	5	4
+----+----------+------+-------+

The resulting DataFrame, displayed through the final `df.show()` command, confirms that the null

values have been successfully replaced. The imputation is complete, and the dataset is now ready for further modeling steps. It is crucial to verify the imputed values against the original data distribution to understand the impact of this cleaning step.

Verification and Analysis of Imputed Values

To verify the successful imputation, let's manually calculate the medians for the two columns of interest from the original dataset (before filling the nulls):

Points Column: The non-null values are . When sorted, the array is . Since there are five values, the median is the third element, which is **8**. We can see that the null value in the `points` column (for Team B) has been correctly replaced with 8.
Assists Column: The non-null values are . When sorted, the array is . The median is the third element, which is **4**. Similarly, the null value in the `assists` column (for Team C) has been correctly replaced with 4.

This verification confirms the accuracy of the PySpark aggregation and imputation process. By using the column median instead of the mean, we ensure that the imputed values are representative of the central tendency, even if the distributions were asymmetrical or contained significant outliers. This technique is especially valuable in environments requiring robust and automated data cleaning pipelines operating on massive scale via PySpark.

Conclusion and Further Reading

Handling missing data is a fundamental aspect of data preparation, and PySpark provides efficient, scalable tools to manage this challenge. By combining the power of the `median` aggregation function and the flexibility of the `fillna()` method, users can implement robust median imputation quickly and reliably across vast DataFrames. This methodology ensures data quality and prepares the dataset optimally for subsequent analytical tasks, reinforcing PySpark's status as a leading tool in the field of distributed computing and big data processing.

Note: You can find the complete documentation for the PySpark `fillna()` function here. Mastering this function, along with other data cleaning tools, is essential for maintaining high-quality big data pipelines.

Related PySpark Data Handling Tutorials

The following tutorials explain how to perform other common data manipulation and cleaning tasks in PySpark, further extending your capability to handle complex datasets:

Tutorial on using PySpark Window Functions for complex aggregation.

Guide to implementing custom User Defined Functions (UDFs) for specialized transformations.

Instructions on efficiently joining large PySpark DataFrames using optimized strategies.

Detailed explanation of handling categorical features using encoders in PySpark MLlib.