How to Calculate Correlation Between Two Columns in PySpark

Authored by

stats writer

February 4, 2026

Analyzing relationships between variables is fundamental to effective data science, especially when dealing with massive datasets characteristic of PySpark environments. Calculating the correlation between two columns allows analysts to quickly quantify the strength and direction of the linear association between them. In the PySpark ecosystem, this critical statistical operation is efficiently handled by specialized functions designed for distributed computing. Understanding how to correctly implement and interpret these calculations is essential for extracting actionable insights from a distributed DataFrame.

This guide provides a comprehensive overview of how to leverage PySpark's built-in statistical capabilities to compute the correlation coefficient, ensuring the analysis is both scalable and statistically sound. We will cover the mechanics of the function, required syntax, and a detailed example using real-world data to clarify the interpretation of the results.

Understanding the Importance of Correlation Coefficients

The correlation coefficient is a standardized measure that ranges strictly between -1 and 1. This numerical value provides crucial context about how two variables move relative to one another. A coefficient approaching 1 signifies a strong **positive correlation**, meaning that as one variable increases, the other tends to increase proportionally. This relationship suggests a strong linear dependency between the two features being analyzed.

Conversely, a coefficient near -1 indicates a strong **negative correlation**, where an increase in one variable is typically associated with a decrease in the other. For example, high training intensity might correlate negatively with injury rates in a poorly managed training program. When the computed coefficient hovers near 0, it suggests that there is little to no **linear relationship** present between the two variables being analyzed. This does not mean the variables are independent, but rather that any relationship present is not linear in nature.

It is paramount to remember that correlation measures only linear relationships; complex non-linear associations may still exist even if the Pearson correlation coefficient is close to zero. Furthermore, correlation does not imply causation, a distinction that must always be maintained during the interpretation phase of data analysis. In the context of big data processed by PySpark, swiftly calculating these coefficients across billions of records saves significant time and computational resources, making it vital for early exploratory data analysis (EDA).

Leveraging the `stat.corr()` Function in PySpark

In PySpark, the primary method for calculating correlation between two columns within a DataFrame is through the use of the `stat.corr()` function. This function is part of the `pyspark.sql.DataFrameStatFunctions` class, which is readily accessible via the `.stat` attribute of any PySpark DataFrame object. This streamlined access allows data scientists to perform

complex statistical operations without needing to convert distributed data into local structures, thereby preserving the benefits of distributed computing.

The required arguments for the `corr()` method are simple yet powerful: the names of the two columns (provided as strings) for which the correlation calculation is desired. By default, PySpark employs the **Pearson correlation coefficient** (PCC), which measures the strength of the linear relationship between normally distributed data. If the data is ordinal or non-normally distributed, an alternative method, such as Spearman, can be explicitly specified as an optional third parameter.

The fundamental syntax for invoking this calculation is remarkably concise, reflecting PySpark's design philosophy of simplifying complex distributed operations. When executed, the function returns a single floating-point number representing the calculated coefficient, ready for immediate statistical interpretation and subsequent use in machine learning feature selection or model diagnostics.

The standard usage pattern for Pearson correlation is as follows, where `df` is the PySpark DataFrame:

df.stat.corr('column_A', 'column_B')

This snippet executes the correlation calculation across the entire distributed dataset, utilizing the optimized cluster resources managed by the Spark session. The result is a highly precise measure between -1.0 and 1.0, quantifying the linear relationship between the data held in `column_A` and `column_B`.

Prerequisites and Setup: Initializing the Data

To execute the correlation calculation effectively, we first require a properly initialized PySpark environment and a populated DataFrame containing numerical data. The initial steps involve importing necessary libraries and establishing a SparkSession, which serves as the fundamental entry point for all PySpark functionality and resource management. It is vital to ensure that the columns targeted for correlation analysis contain numerical data, as correlation is undefined for nominal or categorical variables.

The following code block demonstrates the complete setup process. We define a small dataset containing statistics for basketball players--specifically `assists`, `rebounds`, and `points`--and transform this raw list of lists into a distributed PySpark DataFrame. This example, while intentionally small for demonstration, accurately reflects the structure required for large-scale sports analytics data where correlating player metrics is a routine task.

We use `SparkSession.builder.getOrCreate()` to instantiate or retrieve the Spark context,

providing the necessary infrastructure for distributed computation. Once the session is active, the data list and column names are efficiently converted into a DataFrame using `spark.createDataFrame()`, establishing the necessary schema and structure for statistical analysis.

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()

```
+-------+--------+------+
|assists|rebounds|points|
+-------+--------+------+
| 4| 12| 22|
| 5| 14| 24|
| 5| 13| 26|
| 6| 7| 26|
| 7| 8| 29|
| 8| 8| 32|
| 8| 9| 20|
| 10| 13| 14|
+-------+--------+------+
```

Practical Example: Calculating Correlation Between Assists and Points

With the DataFrame successfully initialized and containing the necessary basketball statistics, we can now proceed to the core task: determining the linear association between the `assists` and `points` columns. This correlation calculation is often used in sports analytics to quantify the relationship between a player's role as a facilitator versus their role as a scorer.

We choose the `assists` and `points` columns specifically to test a common sports hypothesis: whether players who accumulate more assists tend to score fewer points themselves, or if the best players excel at both metrics simultaneously. By invoking the `.stat.corr()` method on the `df` DataFrame and providing the column names as strings, PySpark efficiently calculates the correlation coefficient across the dataset's partitions without requiring manual data shuffling or complex distributed logic.

The following execution block demonstrates the precise syntax required for this operation and shows the resulting output, which is a single floating-point number quantifying the strength and direction of the relationship:

#calculate correlation between assists and points columns
df.stat.corr('assists', 'points')

-0.32957304910500873

The immediate output following the execution of the command is the calculated coefficient, approximately **-0.329573**. This numerical result summarizes the entire relationship between the two variables based on the available data, and the next step is critically important: translating this number into actionable domain knowledge.

Interpreting the Calculated Coefficient (-0.32957)

The computed correlation coefficient of **-0.32957** immediately reveals two fundamental characteristics of the linear relationship between `assists` and `points` in this sample dataset. Firstly, the negative sign is definitive: it indicates a **negative association** between the two variables. Secondly, the magnitude (approximately 0.33) suggests a relationship that is weak to moderate in strength.

A negative association signifies that as the value of one variable increases, the value of the other variable tends to decrease. Specifically, in this context, when a player's number of **assists** increases, their number of **points** tends to decrease. Conversely, players with fewer assists generally tend to have higher point totals. This result supports the general observation in basketball that players often specialize either as primary facilitators (passers) or primary scorers.

It is important not to overstate the strength of this finding. A correlation of -0.33 is not a strong predictive relationship. It suggests that while the trend exists, the relationship is highly influenced by other unaccounted variables. Had the coefficient been closer to -0.9 or -1.0, we could confidently assert that the relationship was nearly perfectly inversely linear. The moderate value indicates that numerous other factors (such as field goal percentage, position, or team strategy) significantly impact individual player statistics.

This rapid insight, gained through the scalable nature of the PySpark calculation, serves as an excellent starting point for deeper investigation. For instance, future analysis might involve segmenting the DataFrame by player position to see if the correlation holds true across different roles on the team.

Choosing the Right Method: Pearson vs. Spearman

While the `df.stat.corr()` function defaults to calculating the Pearson correlation coefficient, PySpark provides the necessary flexibility to calculate alternative measures of association, most importantly the Spearman Rank Correlation. The choice between these methods is critical and depends heavily on the characteristics of the data and the type of relationship being measured.

The **Pearson coefficient** is ideal when variables are measured on an interval or ratio scale, the relationship is known or suspected to be linear, and the data is approximately normally distributed. It is a powerful measure but is highly sensitive to outliers, meaning a single extreme data point can significantly skew the resulting coefficient. If these assumptions are violated, the Pearson result may be misleading.

The **Spearman Rank Correlation**, conversely, is a non-parametric statistic that measures the strength and direction of the monotonic relationship between two variables. It operates on the rank order of the data rather than the raw values. Because it uses ranks, it is much less sensitive to outliers and does not assume a specific distribution or strictly linear relationship, making it robust for skewed data or when only the relative order of values matters.

To calculate the Spearman correlation in PySpark, one simply passes the method name as an optional third argument to the `corr()` function. For example, if we wished to use the Spearman method for our basketball statistics:

df.stat.corr('assists', 'points', 'spearman')

If a data scientist suspects non-linearity, ordinal scaling, or has concerns about heavily skewed data, utilizing the Spearman method provides a robust and reliable alternative measure of association that is particularly valuable in exploratory analysis where underlying distribution assumptions are often uncertain.