How to Round a Date to the First Day of the Week in PySpark

Authored by

stats writer

February 3, 2026

Manipulating temporal data is a core requirement in modern data analysis, particularly when preparing information for aggregation or visualization. A common task involves normalizing dates, such as rounding them to the beginning of a specific time period. When working with the distributed computing framework PySpark, developers often need an efficient way to round dates to the first day of the corresponding week. Although one traditional method involves calculating the specific day of the week using the 'dayofweek' function and then applying an offset via 'date_add' or 'date_sub', the most straightforward and performant technique leverages a specialized SQL function built directly into PySpark's toolkit. This method streamlines the process of ensuring that all data points within the same seven-day period map back to a consistent starting date, which is crucial for accurate weekly summaries.

To round a date to the first day of the week in PySpark, one could theoretically use the 'dayofweek' function to determine the day of the week for a given date. Then, one would subtract the necessary number of days (the day index minus one, assuming Monday=1) from the original date to get the first day of the week. This complex logic can be achieved by utilizing the 'date_add' or 'date_sub' functions in PySpark, specifying the number of days to be added or subtracted dynamically based on the day index. While feasible, this approach requires conditional logic and multiple function calls. Fortunately, PySpark offers a much cleaner, single-function solution specifically designed for this type of date truncation, greatly simplifying implementation and improving readability for robust analysis, especially when working with high-volume weekly data.

PySpark: Round Date to First Day of Week

The Importance of Weekly Date Normalization in DataFrames

In fields relying on temporal data, such as finance, logistics, or sales reporting, data is often aggregated and analyzed weekly. Before aggregation can occur, it is paramount that all individual dates within a seven-day period are mapped back to a single, consistent identifier, typically the start date of that week. If this normalization is not performed, attempts to group or summarize data will result in incorrect counts or incomplete summaries across weekly boundaries. This normalization process ensures that derived metrics, such as weekly sales volume or weekly average usage, are accurately calculated across the entire DataFrame.

The choice of which day constitutes the "first day" of the week is critical and often depends on business or regional conventions. For instance, the ISO 8601 standard dictates Monday as the start of the week, a convention widely adopted in many parts of the world and often favored by computational frameworks. Conversely, some regions, particularly the United States, commonly define Sunday as the start of the week. Understanding the default behavior of the specific tool being used--in this case, PySpark--is essential to guarantee the resulting aggregated data aligns with the intended analytical scope.

PySpark, designed for processing vast datasets efficiently, provides optimized functions to handle these date manipulations at scale. Relying on built-in functions like `trunc` ensures that the processing is pushed down to the underlying Spark engine, maximizing performance and avoiding expensive User Defined Functions (UDFs) or complex conditional logic that could slow down large-scale transformations. This efficiency is a core reason why leveraging optimized functions is considered best practice when performing time series preparation within a distributed environment.

Implementing the Efficient Solution: PySpark's `trunc` Function

PySpark's SQL function module offers a dedicated function, `trunc`, which serves to truncate a date or timestamp to the beginning of the specified unit of measure. This function is incredibly versatile, supporting truncation by year ('YYYY'), month ('MM'), or, relevant to our goal, by week ('week'). By designating 'week' as the format unit, the trunc function automatically calculates and returns the date corresponding to the start of the week containing the input date.

This operation is performed across the entire DataFrame column in a parallelized manner, ensuring high throughput. The syntax is straightforward, requiring only the name of the date column and the keyword 'week'. It eliminates the necessity for manual calculations involving day indices or offsets, thereby reducing potential errors and simplifying the codebase dramatically.

You can use the following syntax to round dates to the first day of the week in a PySpark DataFrame:

import pyspark.sql.functions as F

#add new column that rounds date to first day of week
df_new = df.withColumn('first_day_of_week', F.trunc('date', 'week'))

This particular example creates a new column named **first_day_of_week** that rounds each date in the **date** column to the first day of the week. Note the use of the alias `F` for `pyspark.sql.functions`, which is standard practice for concise access to Spark's built-in SQL functions. The result will be a new column containing date type values, all aligned to the Monday of their respective weeks, based on Spark's default week definition.

Setting Up the Environment and Sample Data

To demonstrate the practical application of the trunc function, we must first establish a running SparkSession and load a sample dataset. The SparkSession acts as the entry point to utilizing Spark functionality, managing the resources and configurations necessary for distributed computation. Once initialized, we can define a simple dataset containing date and corresponding sales information, simulating real-world transaction data that requires weekly aggregation.

The subsequent example shows how to utilize this syntax in a complete, practical scenario. Suppose we have the following PySpark DataFrame that contains information about the sales made on various days at some company. This data is purposely sparse, spanning multiple months and quarters, to showcase how the truncation accurately handles different week boundaries throughout the year.

The preparation steps involve importing the necessary components, defining the structure of the data (the columns), and then initializing the DataFrame itself. This structured approach ensures reproducibility and clarity in data transformation exercises.

Example: How to Round Date to First Day of Week in PySpark

The following code block initializes the Spark session, defines the input data, and displays the original DataFrame. This step confirms the starting point before any transformations are applied, allowing us to verify the output against the expected weekly truncation logic.

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()

+----------+-----+
| date|sales|
+----------+-----+

|2023-04-11| 22|
|2023-04-15| 14|
|2023-04-17| 12|
|2023-05-21| 15|
|2023-05-23| 30|
|2023-10-26| 45|
|2023-10-28| 32|
|2023-10-29| 47|
+----------+-----+

Suppose we would like to round each date in the **date** column to the first day of the week. This transformation prepares the data for subsequent grouping operations, enabling the calculation of total weekly sales.

Applying the Truncation and Analyzing the Output

We can now use the following syntax, importing the necessary functions module and applying the `trunc` operation directly onto the 'date' column. The result is stored in `df_new`, demonstrating the immutability of PySpark DataFrames; the original `df` remains unchanged, and a new DataFrame containing the added column is created.

The application of `F.trunc('date', 'week')` is highly efficient. It instructs the Spark engine to perform the date calculation in parallel across all partitions of the data, ensuring scalability even for petabyte-scale datasets. This efficiency is critical for maintaining performance in large-scale data analysis pipelines.

We can use the following syntax to do so:

import pyspark.sql.functions as F

#add new column that rounds date to first day of week
df_new = df.withColumn('first_day_of_week', F.trunc('date', 'week'))

#view new DataFrame
df_new.show()

+----------+-----+-----------------+
| date|sales|first_day_of_week|
+----------+-----+-----------------+
|2023-04-11| 22| 2023-04-10|
|2023-04-15| 14| 2023-04-10|

|2023-04-17| 12| 2023-04-17|
|2023-05-21| 15| 2023-05-15|
|2023-05-23| 30| 2023-05-22|
|2023-10-26| 45| 2023-10-23|
|2023-10-28| 32| 2023-10-23|
|2023-10-29| 47| 2023-10-23|
+----------+-----+----------------+

The new **first_day_of_week** column contains each date from the **date** column rounded to the first day of the week. This column can now be used as the primary key for grouping operations to calculate weekly summaries.

Understanding PySpark's Default Week Start (Monday)

It is crucial to understand that when using `F.trunc('date', 'week')` in PySpark, the framework defaults to the ISO 8601 standard, where the "first" day of the week is considered to be **Monday**. This behavior is standard across many computing environments and aligns with international business practices. If your organizational standard requires a Sunday start, you must employ an alternative method involving date arithmetic, as discussed below.

For example, by reviewing the results:

The date **2023-04-11** (a Tuesday) falls in the week beginning Monday, **2023-04-10**. Thus, its truncated date is **2023-04-10**.

Similarly, **2023-04-15** (a Saturday) is in the same Monday-starting week, hence it also maps to **2023-04-10**.

The date **2023-04-17** (a Monday) is already the first day of its week, so it truncates to itself, **2023-04-17**.

The dates **2023-10-26**, **2023-10-28**, and **2023-10-29** all fall within the week that starts on Monday, October 23, 2023, and are therefore correctly mapped to **2023-10-23**.

This confirmation ensures that the truncation logic is working as intended according to the ISO standard. If a different week start is required, the analyst must adjust the methodology.

Alternative Approach: Customizing Week Start Using Date Arithmetic

While the trunc function is efficient, its fixed Monday start may not suit all organizational needs. If the requirement is to use Sunday as the first day of the week, analysts must revert to using date

arithmetic involving the 'dayofweek' function (which returns 1 for Sunday and 7 for Saturday) in conjunction with `date_sub`.

To implement a Sunday-based week start, one determines the numerical representation of the day (e.g., Sunday=1, Monday=2, ..., Saturday=7). We then subtract `(dayofweek() - 1)` days from the current date. This subtraction guarantees that the resulting date is the Sunday of the current week. This method, although requiring slightly more complex syntax, offers the flexibility necessary for custom temporal alignment in time series analysis.

For instance, to achieve Sunday truncation: `df.withColumn('sunday_start', F.date_sub(F.col('date'), F.dayofweek('date') - 1))`. This formula provides full control over the definition of the week boundary, which is vital when merging data from systems that adhere to non-ISO weekly standards. However, it is essential to remember that such custom logic might be slightly less performant than the optimized, native SQL `trunc` function, especially on massive datasets.

Conclusion and Best Practices for Time Series Analysis

Rounding dates to the first day of the week is a fundamental step in preparing data for weekly data analysis and reporting. PySpark provides the highly optimized `F.trunc('date', 'week')` function, offering the simplest and most performant solution for this task, defaulting to the Monday (ISO 8601) week start. Analysts should prioritize this function unless specific business requirements dictate a non-Monday start day.

When a custom week start is necessary, the alternative method using `F.dayofweek` and `F.date_sub` offers the necessary flexibility. Regardless of the chosen method, thoroughly validating the output against known dates is a necessary best practice to ensure the weekly aggregation boundaries are correctly defined for the downstream analytical tasks. For further technical details on the function's parameters and behavior, it is recommended to consult the official PySpark documentation.

Note: You can find the complete documentation for the PySpark **trunc** function online.