How to Find the Day of the Week in PySpark

Authored by

stats writer

February 3, 2026

PySpark is recognized as an indispensable, high-performance tool for data analysis, offering scalable solutions for manipulating massive datasets through the power of Python. A foundational requirement in virtually all time-series and event-based analyses involves extracting temporal attributes, such as determining the specific day of the week associated with a given date column. Successfully accomplishing this task within the distributed environment of Spark requires utilizing specialized functions provided by the PySpark SQL module, ensuring both efficiency and accuracy across diverse data partitions.

The core functionality for date extraction in PySpark is handled primarily by the built-in functions library, which allows data engineers and analysts to transform date columns without resorting to complex user-defined functions (UDFs). Specifically, the `dayofweek` function offers a direct, standardized approach to convert a date entry into its numerical representation of the day of the week. This straightforward integration capability means that temporal feature engineering can be seamlessly incorporated into larger data processing pipelines, crucial for building robust models or generating time-based aggregations.

Furthermore, understanding how PySpark handles date and time operations is vital for maintaining data integrity. The `dayofweek` function inherently accounts for specific nuances like locale and time zone settings, which is essential when dealing with global or geographically diverse datasets. This built-in reliability ensures that the resulting day of the week attribution is consistently accurate, mitigating common issues related to time zone misalignment that often plague manual date conversion methods in distributed systems.

Mastering Day of Week Extraction in PySpark: A Comprehensive Guide

The Core Functions for Day of Week Extraction

To effectively extract the day of the week from a date column within a DataFrame, PySpark provides two primary functions. The first is the native `dayofweek` function, which returns an integer representing the day. The second is the highly versatile `date_format` function, which leverages specific pattern strings to return the

day as a name (either abbreviated or full). Choosing the correct function depends entirely on the required output format--whether numerical indexing is needed for further computation or textual names are required for reporting and visualization purposes.

The core challenge often lies in the numerical representation, as different systems define the start of the week differently (Sunday=1 vs. Monday=1, following the ISO 8601 standard). While the `dayofweek` function defaults to the standard where Sunday is 1 and Saturday is 7, the flexibility of PySpark allows for easy mathematical transformation to align with the ISO standard or any other required convention, ensuring maximum adaptability for international projects.

Below, we detail the four most common and essential methods available for obtaining the day of the week from date columns in a PySpark DataFrame. These methods cover both numerical and textual outputs, providing a complete toolkit for any date manipulation requirement encountered during large-scale data processing.

Method 1: Get Day of Week as Number (Sunday = 1)

```
import pyspark.sql.functions as F

df_new = df.withColumn('day_of_week', F.dayofweek('date'))
```

Method 2: Get Day of Week as Number (Monday = 1)

```
import pyspark.sql.functions as F

df_new = df.withColumn('day_of_week', ((F.dayofweek('date')+5)%7)+1)
```

Method 3: Get Day of Week as Abbreviated Name (e.g., Mon)

```
import pyspark.sql.functions as F

df_new = df.withColumn('day_of_week', F.date_format('date', 'E'))
```

Method 4: Get Day of Week as Full Name (e.g., Monday)

```
import pyspark.sql.functions as F
```

```
df_new = df.withColumn('day_of_week', F.date_format('date', 'EEEE'))
```

Setting Up the PySpark Environment and Sample Data

Before demonstrating the practical application of these functions, it is essential to establish a working PySpark context and define a sample DataFrame that includes a date column. This standardized setup ensures that all subsequent examples are reproducible and clearly illustrate the output of each method. We begin by initializing a `SparkSession`, which serves as the entry point to programming Spark with the Dataset and DataFrame API.

Our example dataset is a simple collection of sales records, featuring a date and the corresponding sales quantity for that day. This structure is highly representative of real-world datasets where temporal analysis is frequently required. Defining the data structure explicitly allows us to control the input, ensuring that we test the functions against known dates to verify the accuracy of the resulting day-of-week extraction.

The following block demonstrates the standard code required to instantiate the session, define the data array, assign column names, and finally, create and display the resulting DataFrame named `df`. This serves as the baseline data for all subsequent transformation examples shown in this guide.

```
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|
+----------+-----+
```


Method 1: Extracting the Day of Week as a Number (Sunday = 1)

The default behavior of the `dayofweek` function aligns with the convention where Sunday is considered the first day of the week, assigned the numerical value 1, and the sequence progresses until Saturday, which is assigned 7. This is the simplest and most direct method for numerical extraction, requiring only a single function call applied to the target date column.

This approach is particularly useful when the numerical output is intended for aggregation or sorting based on a Sunday-start week structure, such as in analyses common in certain financial or cultural reporting contexts. We use `df.withColumn` to create a new column, `day_of_week`, populated with the calculated integer value.

The syntax below demonstrates this implementation, confirming that `2023-05-21`, which is a Sunday, correctly receives the output value of 1, while `2023-04-17`, a Monday, receives 2, following the internal PySpark convention.

```
import pyspark.sql.functions as F

#add new column that displays day of week as number (Sunday=1)
df_new = df.withColumn('day_of_week', F.dayofweek('date'))

#view new DataFrame
df_new.show()

+----------+-----+-----------+
| date|sales|day_of_week|
+----------+-----+-----------+
```

```
|2023-04-11| 22| 3|
|2023-04-15| 14| 7|
|2023-04-17| 12| 2|
|2023-05-21| 15| 1|
|2023-05-23| 30| 3|
|2023-10-26| 45| 5|
|2023-10-28| 32| 7|
|2023-10-29| 47| 1|
+----------+-----+-----------+
```

Method 2: Adjusting for ISO Standard (Monday = 1)

In many global and technical contexts, particularly those adhering to the ISO 8601 standard, Monday is defined as the first day of the week (1), and Sunday is the last (7). Since PySpark's native `dayofweek` function does not inherently follow this standard, a mathematical transformation is required to shift the numerical index. This adjustment ensures compatibility when integrating PySpark results with systems that mandate the ISO convention.

The transformation formula `((F.dayofweek('date') + 5) % 7) + 1` is utilized to achieve this shift. By adding 5, we effectively rotate the week index. The modulo 7 operation handles

the wrap-around from Sunday (originally 1) back to 7 (the end of the week). Finally, adding 1 ensures the result is a 1-based index (Monday=1) rather than a 0-based index.

This complex but critical calculation allows us to produce an output where Monday is correctly indexed as 1, Tuesday as 2, and Sunday as 7. This tailored output is essential for business logic that relies on a Monday start, such as calculating week numbers or analyzing work-week performance metrics.

```
import pyspark.sql.functions as F

#add new column that displays day of week as number (Monday=1)
df_new = df.withColumn('day_of_week', ((F.dayofweek('date')+5)%7)+1)

#view new DataFrame
df_new.show()

+----------+-----+-----------+
| date|sales|day_of_week|
+----------+-----+-----------+
```

```
|2023-04-11| 22| 2|
|2023-04-15| 14| 6|
|2023-04-17| 12| 1|
|2023-05-21| 15| 7|
|2023-05-23| 30| 2|
|2023-10-26| 45| 4|
|2023-10-28| 32| 6|
|2023-10-29| 47| 7|
+----------+-----+-----------+
```

Method 3: Obtaining the Abbreviated Day Name

When the output format requires textual representation rather than a numerical index, the `date_format` function becomes the preferred tool. This highly flexible function accepts a date column and a standard format string to dictate the desired output structure. To retrieve the abbreviated name of the day of the week (e.g., "Mon," "Tue," "Sun"), we use the format pattern `'E'`.

Using `'E'` is advantageous for reporting dashboards or visualizations where space is limited, but readability is still crucial. The function inherently handles the localization of the abbreviation based on the configured environment settings, ensuring that the results are

contextually appropriate.

The following code snippet demonstrates the implementation, showcasing how `2023-04-17` (Monday) is successfully transformed into "Mon," providing a clear and concise textual summary of the temporal data.

```
import pyspark.sql.functions as F

#add new column that displays day of week as abbreviated name
df_new = df.withColumn('day_of_week', F.date_format('date', 'E'))

#view new DataFrame
df_new.show()

+----------+-----+-----------+
| date|sales|day_of_week|
+----------+-----+-----------+
|2023-04-11| 22| Tue|
|2023-04-15| 14| Sat|
|2023-04-17| 12| Mon|
|2023-05-21| 15| Sun|
|2023-05-23| 30| Tue|
```

|2023-10-26| 45| Thu|
|2023-10-28| 32| Sat|
|2023-10-29| 47| Sun|
+----------+-----+-----------+

Method 4: Retrieving the Full Day Name

For scenarios requiring the complete, unabbreviated name of the day of the week (e.g., "Monday," "Tuesday," "Sunday"), we again rely on the powerful `date_format` function, but this time utilizing the format pattern `'EEEE'`. The repetition of the format character `E` signals to Spark that the full name should be returned, offering the highest level of detail for textual date representation.

This approach is ideal for final reporting, internal documentation, or when integrating data with systems that require verbose labels for temporal data. The full name format ensures maximum clarity, preventing any ambiguity that might arise from abbreviations, especially in multilingual or highly formal documentation.

The output generated by applying the `'EEEE'` pattern

clearly displays the full day name for each record, confirming the functional transformation. This method provides the most readable output for human consumption, making it a critical tool in the final stages of a data pipeline focused on presentation layer data.

```
import pyspark.sql.functions as F

#add new column that displays day of week as full name
df_new = df.withColumn('day_of_week', F.date_format('date', 'EEEE'))

#view new DataFrame
df_new.show()

+----------+-----+-----------+
| date|sales|day_of_week|
+----------+-----+-----------+
|2023-04-11| 22| Tuesday|
|2023-04-15| 14| Saturday|
|2023-04-17| 12| Monday|
|2023-05-21| 15| Sunday|
|2023-05-23| 30| Tuesday|
|2023-10-26| 45| Thursday|
```

|2023-10-28| 32| Saturday|
|2023-10-29| 47| Sunday|
+----------+-----+-----------+

Summary of PySpark Date Formatting Patterns

To ensure analysts can leverage the full capability of the `date_format` function beyond just day extraction, it is helpful to list key format patterns derived from the standard Java SimpleDateFormat, which PySpark utilizes. Understanding these patterns allows for precise control over how temporal data is presented and consumed in subsequent steps.

These patterns are crucial not only for day-of-week extraction but for formatting dates into custom strings, often necessary when exporting data to legacy systems or generating human-readable reports that follow specific date conventions.

The following ordered list summarizes several vital pattern symbols used within PySpark date functions:

Y/y: Year. Use `y` for the year (e.g., 2023) or `yy` for the last two digits (e.g., 23).

M: Month. Use `M` for month number, `MMM` for abbreviated month name (e.g., Apr), and `MMMM` for the full month name (e.g., April).

D: Day in Year. Returns the day number within the current year (1-366).

d: Day in Month. Returns the day number within the month (1-31).

E: Day in Week. Use `E` for abbreviated day name (Mon) and `EEEE` for the full day name (Monday).

a: Am/Pm Marker. Used to denote morning or afternoon time divisions.

H: Hour (0-23). Military time format.

h: Hour (1-12). Standard clock format.

Conclusion and Further Resources

Extracting the day of the week is a fundamental operation in PySpark, easily achieved through the native `dayofweek` function for numerical outputs or the versatile `date_format` function for textual results. By mastering these four methods--Sunday-based

numbering, ISO-compliant numbering, abbreviated names, and full names--data professionals can efficiently preprocess temporal data for complex analytical models or clear reporting.

The ability to quickly toggle between numerical standards (Sunday=1 versus Monday=1) using simple yet effective mathematical transformations highlights the robust flexibility of **PySpark** SQL functions. This adaptability is critical when processing data derived from varied international systems or meeting strict client reporting standards.

For those seeking to expand their proficiency in temporal data manipulation within Spark, investigating other functions related to dates, such as `weekofyear`, `date_add`, and `months_between`, is highly recommended. These tools collectively form the foundation for advanced time-series analysis on large-scale datasets.

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