How to Find the Day of the Week in PySpark DataFrames

Authored by

stats writer

January 18, 2026

Introduction to Date Manipulation in PySpark

Processing time-series and date-based data is a fundamental requirement in modern data engineering and analytics. When working with large-scale datasets, the PySpark (1/5) framework provides robust and optimized methods for handling complex transformations. One common task is extracting the day of the week from a date column within a DataFrame (1/5). Understanding the specific functions available allows data scientists to standardize workflows, apply time-based filtering, and conduct deep temporal analysis efficiently. This guide details four distinct and authoritative methods for determining the day of the week for dates stored in a PySpark DataFrame, covering both numeric and string representations.

The choice of method often depends on the desired output format and regional standards (e.g., whether the week begins on Sunday or Monday). PySpark offers built-in functions within the pyspark.sql.functions (1/5) module that simplify this process dramatically, eliminating the need for complex User-Defined Functions (UDFs) for basic date logic. Mastering these functions is essential for anyone utilizing Spark for big data processing, as efficient date handling directly impacts processing speed and resource usage across the cluster.

We can employ several specialized functions provided by the PySpark framework to extract the day of the week. These methods range from simple numerical extraction to generating localized, formatted strings. Below is a concise overview of the primary functions used, followed by detailed, practical examples demonstrating their implementation:

Method 1: Get Day of Week as Number (Sunday =1) using `F.dayofweek()`

import pyspark.sql.functions as F

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

Method 2: Get Day of Week as Number (Adjusted for 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) using `F.date_format()`

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) using `F.date_format()`

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 diving into the specific transformations, we must establish a working PySpark session and create a sample DataFrame (2/5) containing date values. This sample data simulates a common scenario in data analysis, where records are associated with specific dates (e.g., sales figures over time). We utilize the `SparkSession` entry point to initialize the environment, ensuring that the necessary distributed computing context is established.

Our example DataFrame, named `df`, consists of two primary columns: `'date'`, which holds the date strings we intend to analyze, and `'sales'`, representing an arbitrary metric associated with that date. It is crucial that the date column is correctly formatted, although PySpark is generally flexible, handling standard 'YYYY-MM-DD' strings effectively when applying date functions.

The following code snippet demonstrates the creation and display of this foundational dataset, which will be used consistently across all subsequent examples to illustrate the output of each day-of-week extraction method. Pay close attention to how the dates are defined and structured, as this forms the input for all our transformations.

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

# Define sample data containing dates and sales figures
data = ,
,
,
,
,
,
,
]

Define descriptive column names
columns =

Create the PySpark DataFrame

df = spark.createDataFrame(data, columns)

# Display the initial DataFrame structure and content
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|
+----------+-----+
```

Utilizing dayofweek() for Numeric Representation (Sunday = 1)

The most direct function for extracting the day of the week as a number is `F.dayofweek()` (1/5). This function is part of the `pyspark.sql.functions` library and is designed to return an integer representing the day, based on the standard SQL convention where **Sunday** is assigned the value of 1, and Saturday is assigned 7. This convention is highly prevalent in various analytical databases and systems, making `dayofweek()` a reliable choice when integrating with such environments.

When applying this function, we use the `.withColumn()` transformation on our existing DataFrame, specifying the new column name (`'day_of_week'`) and the calculation applied to the target date column (`'date'`). It is important to remember this standard definition: Sunday (1), Monday (2), Tuesday (3), Wednesday (4), Thursday (5), Friday (6), and Saturday (7). Data analysts often use this numeric output for grouping weekend days (1 or 7) versus weekdays (2 through 6).

The resulting output clearly shows the transformation applied to each row, providing a simple, quantifiable representation of the date's position within the week cycle. For instance, the date '2023-05-21', which is a Sunday, correctly yields the value 1.

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 the resulting 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|
+----------+-----+-----------+
```

Adjusting Numeric Output for Monday Start of Week (ISO Standard)

While the default behavior of `F.dayofweek()` (2/5) sets Sunday as 1, many countries and international standards, such as the ISO 8601 standard, define **Monday** as the start of the week (Monday = 1, Sunday = 7). If your business logic or target system requires this ISO-compliant numbering, a slight arithmetic manipulation is necessary on the result provided by the native function.

This adjustment relies on simple modular arithmetic. Since `dayofweek()` returns 1 for Sunday, we need to shift this value so that Sunday (1) becomes 7, and Monday (2) becomes 1. The formula `((F.dayofweek(date) + 5) % 7) + 1` achieves this precise rotation. We add 5, perform a modulo 7 operation to handle the wrap-around, and finally add 1 to ensure the resulting sequence starts from 1 instead of 0. This calculation is performed directly within the `.withColumn()` function, allowing the computation to be executed efficiently across the Spark cluster.

This approach is particularly useful in environments prioritizing ISO standards for data interoperability. For instance, '2023-04-17', which is a Monday, now correctly returns 1, while '2023-05-21', a Sunday, returns 7, aligning perfectly with the Monday-start convention. This flexibility demonstrates the power of combining built-in functions with standard arithmetic operations within PySpark (2/5) transformations.

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 the resulting 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|
+----------+-----+-----------+
```

Formatting Dates Using date_format() for Abbreviated Names

While numerical representation is useful for internal processing, presenting the day of the week as a recognizable name often improves readability for end-users or reporting dashboards. `F.date_format()` (1/5) is the function designed for this purpose within pyspark.sql.functions (2/5). This powerful function accepts two key arguments: the date column reference and a standard formatting pattern string.

To obtain the abbreviated day name (e.g., Mon, Tue, Sat), we use the format specifier `'E'`. This specifier is part of the common SQL and Java date formatting standards that Spark adheres to. Applying `F.date_format(col('date'), 'E')` transforms the date object into a string containing the three-letter abbreviation of the corresponding day. This output is ideal when screen space is limited or when a concise presentation is preferred, such as in table headers or condensed summaries.

It is crucial to note that the capitalization and specific abbreviation format are typically dependent on the underlying Spark configuration and the locale setting of the execution environment, although standard English abbreviations are the default behavior. The example below showcases how dates like '2023-04-11' (Tuesday) are correctly converted to 'Tue'.

import pyspark.sql.functions as F

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

# View the resulting 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|
+----------+-----+-----------+
```

Retrieving the Full Day Name via Date Formatting

For scenarios requiring maximum clarity, such as formal reports or detailed logging, displaying the full name of the day (e.g., Monday, Tuesday) is the preferred method. Utilizing the same `F.date_format()` (2/5) function, we simply adjust the format specifier string. Instead of a single 'E', we use the four 'E's format: `'EEEE'`.

The pattern `'EEEE'` instructs Spark to return the full, unabbreviated name of the day corresponding to the input date. This capability ensures that the output column is easily understandable by any user, regardless of their familiarity with date abbreviations. This string output can then be readily used for filtering operations or joining with other dimension tables where textual day names might be stored.

The ability to switch easily between numeric, abbreviated, and full string formats using just two primary functions--`F.dayofweek()` (3/5) and `F.date_format()` (3/5)--showcases the efficiency of the pyspark.sql.functions (3/5) library. Below, observe how the date '2023-10-26', which was previously 'Thu' or '5', is now rendered as 'Thursday'.

```
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 the resulting 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|
+----------+-----+-----------+
```

Advanced Considerations: Localization and Format Codes

When dealing with global data, developers must consider the implications of localization. The output of `F.date_format()` (4/5) for day names (both 'E' and 'EEEE') is influenced by the current session's locale settings. If the Spark context is configured for a non-English locale (e.g., German or French), the resulting day names will be localized accordingly. It is vital to manage this configuration if consistency in output language is required across different processing environments.

Furthermore, the format codes used with `date_format()` follow the conventions derived from the Java SimpleDateFormat pattern. While 'E' and 'EEEE' specifically handle day names, many other codes exist for extracting different date components, such as `'w'` for week of year, `'M'` for month, and `'y'` for year. Understanding the full range of these codes allows for flexible and complex date mask application, crucial for detailed feature engineering in predictive modeling tasks.

For projects that need to handle time zones explicitly, PySpark (3/5) also offers functions like `from_utc_timestamp` and `to_utc_timestamp`. Although the day of the week determination is generally straightforward relative to the date itself, confirming the time zone interpretation of the source data prevents common off-by-one errors that can occur around midnight, particularly when dealing with dates near daylight savings time transitions or across major regional boundaries.

Summary of PySpark Date Functions and Use Cases

We have explored four primary, efficient methods for extracting the day of the week from date

columns within a DataFrame (3/5) using native PySpark (4/5) functions. Whether the requirement is numerical output for programmatic use (using `F.dayofweek()` (4/5)), or descriptive string output for reporting (using `F.date_format()` (5/5)), the framework provides optimized tools within pyspark.sql.functions (4/5). These methods are far superior in performance compared to applying similar logic through lambda functions or UDFs, as they leverage Spark's Catalyst optimizer.

Choosing the right method streamlines data preparation. For analytical tasks such as calculating average sales per day type (weekday vs. weekend), the numerical methods (1 and 2) are most appropriate. Conversely, for presentation layer preparation, the formatting methods (3 and 4) ensure human readability. Always prioritize the use of built-in functions over custom code to maximize processing efficiency on large clusters.

These functions form a crucial foundation for any serious time-series analysis in the Spark ecosystem. By mastering date extraction and formatting, developers can unlock deeper insights into temporal data patterns, leading to more informed business decisions.

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