Learning to Extract Weekdays from Dates Using R and the Lubridate Package

Authored by

Mohammed looti

November 2, 2025

Determining the day of the week from a given date field is a foundational requirement across numerous data analysis and business intelligence tasks. Whether segmenting sales data by weekday or scheduling automated reports, accurately extracting this temporal dimension is crucial. Within the widely used R programming environment, the most modern, efficient, and reliable methodology for handling date and time objects, including the precise extraction of weekdays, is achieved by leveraging the comprehensive functions provided by the specialized lubridate package. This package drastically simplifies complex date operations that often prove cumbersome when relying solely on base R functions, providing intuitive syntax and robust handling of various date formats.

The core utility for this specific task is the highly flexible `wday()` function, which is expertly designed to isolate and format the day of the week rapidly. Its adaptability ensures that analysts can tailor the output precisely to their requirements--whether the need is for a simple numeric index for sorting, or a fully descriptive character label suitable for direct presentation in reports. The versatility of `wday()` stems from its ability to handle four primary modes of operation, allowing for seamless integration into diverse analytical pipelines and international reporting standards. Understanding these modes is key to mastering date manipulation in R.

We will delve into these four primary configurations of the **`wday()`** function, showcasing how each modification fundamentally alters the output format, thereby allowing you to choose the perfect representation for your specific data analysis needs:

Method 1: Numeric Index (Sunday Start). This is the default setting, returning an integer (1-7) where the traditional North American week structure dictates that 1 represents Sunday.
Method 2: Numeric Index (Monday Start). By adjusting the function's arguments, the numbering can be aligned with international standards such as ISO 8601, where 1 corresponds to Monday.
Method 3: Character Labels (Abbreviated). This mode returns clear, three-letter character labels (e.g., Mon, Tue), enhancing readability without consuming excessive display space.
Method 4: Character Labels (Full Names). The most user-friendly presentation, providing the complete name of the day (e.g., Monday, Tuesday), ideal for final reporting.

The subsequent examples provide a quick reference guide to the necessary syntax for each method, demonstrating the conciseness and power of the **`wday()`** function provided by the lubridate package.

Method 1: Find Numeric Day of Week (Assuming Week Starts on Sunday)

wday(df$date_column)

Method 2: Find Numeric Day of Week (Assuming Week Starts on Monday)

wday(df$date_column, week_start=1)

Method 3: Find Character Day of Week (Using Abbreviated Labels)

wday(df$date_column, label=TRUE)

Method 4: Find Character Day of Week (Using Full Weekday Labels)

wday(df$date_column, label=TRUE, abbr=FALSE)

Setting Up the Environment and Sample Data

Before we can execute the practical demonstrations of the four **`wday()`** methods, it is essential to prepare our R environment. This involves two critical steps: loading the necessary external package and ensuring we have a clean, structured dataset for demonstration. We must explicitly load the lubridate package using the standard `library()` function, making all its powerful date and time manipulation tools available for use in the current session.

For the purpose of clear illustration, we will utilize a simple, yet representative, data frame named **`df`**. This structure will contain a column of dates and corresponding sales figures. Using a data frame allows us to simulate a real-world scenario where analysts often need to append calculated temporal metrics, such as the day of the week, directly back into their primary dataset for subsequent grouping or filtering operations. The integrity of the process hinges on the date column being correctly recognized as a Date or POSIXct object.

A significant advantage of the lubridate package is its intelligent handling of common date strings. Although our sample dates are initially defined as character strings in the standard YYYY-MM-DD format, the `wday()` function is robust enough to parse and convert these strings internally. This eliminates the need for explicit, manual type casting in many common scenarios, streamlining the data preparation workflow considerably. The following code block executes the setup, initializing the environment and displaying the newly created sample structure:

library(lubridate)

#create data frame
df <- data.frame(date=c('2020-10-11', '2020-10-19', '2020-10-31'),
sales=c(435, 768, 945))

#view data frame
df

date sales
1 2020-10-11 435
2 2020-10-19 768
3 2020-10-31 945

Method 1: Obtaining the Numeric Day (Sunday Start Default)

The default execution of the `wday()` function is designed to return an integer value between 1 and 7, representing the day of the week based on a specific, commonly used convention. By default, the function assumes the week commences on Sunday. This structure is prevalent in various reporting standards, particularly within the United States, and is often the expected output when integrating with legacy systems or certain database structures.

Under this default setting, the numeric indices are mapped linearly: 1 corresponds to Sunday, 2 to Monday, continuing sequentially until 7, which signifies Saturday. Analysts frequently employ this numeric output when the goal is to perform mathematical operations, calculate weekly averages, or sort data based on the temporal sequence of days, irrespective of the character name. Since no additional arguments are required to invoke this behavior, it is the simplest form of the function call.

We demonstrate this by applying the default **`wday()`** function directly to the **`date`** column of our sample data frame, **`df`**. The resulting integer sequence, representing the day of the week, is stored in a new column aptly named **`weekday`**. The subsequent code block illustrates the application and the resulting modification to the dataset, confirming the numeric mapping based on the Sunday start convention:

#find day of week
df$weekday <- wday(df$date)

#view updated data frame
df

date sales weekday
1 2020-10-11 435 1
2 2020-10-19 768 2
3 2020-10-31 945 7

Observing the results, we can interpret the mapping clearly: the date 2020-10-11, which falls on a Sunday, is correctly assigned the value 1. Similarly, 2020-10-19, being a Monday, receives the value 2, and 2020-10-31, a Saturday, is indexed as 7. This foundational understanding of the default index mapping is crucial for accurate interpretation of numeric day-of-week analysis.

Method 2: Adjusting the Week Start to Monday

While the Sunday start convention is common in certain contexts, much of the world, along with international business and technical standards, defines the week as commencing on Monday. To ensure data analysis complies with global reporting standards, notably the widely adopted ISO 8601 standard for date and time data, we must explicitly modify the behavior of the `wday()` function. This modification is achieved by supplying the **`week_start`** argument.

By specifying **`week_start=1`** (where 1 internally corresponds to Monday in the lubridate context when defining the starting day), we instruct the lubridate package to re-index the numeric output. Consequently, Monday is assigned the number 1, Tuesday becomes 2, and this sequence continues until Sunday, which is assigned the value 7. This setup is indispensable when performing analysis that requires strict adherence to international or European conventions, ensuring consistency and preventing misinterpretation of temporal data.

Implementing the **`week_start`** parameter is straightforward and significantly impacts the numeric output, providing essential flexibility for multi-national data projects. This is particularly valuable in time series analysis where weekly cycles are paramount and must conform to a standardized start day. Below is the implementation demonstrating how the **`week_start=1`** argument shifts the indexing structure:

#find day of week
df$weekday <- wday(df$date, week_start=1)

#view updated data frame
df

date sales weekday
1 2020-10-11 435 7
2 2020-10-19 768 1
3 2020-10-31 945 6

As confirmed by the updated output, the date 2020-10-19, which is a Monday, now correctly receives the index value of 1. Conversely, the date 2020-10-11, now recognized as the last day of the week in this standard, is assigned the value 7. This configuration is highly recommended for all analyses that mandate a Monday start, aligning the results with the principles of ISO 8601.

Method 3: Returning Abbreviated Day Names (Character Labels)

While numeric codes are excellent for computational efficiency and sorting, they often lack the immediate clarity needed for human-readable reports and visualizations. To transition the output

from an integer index to descriptive character labels, we employ the `label=TRUE` argument within the `wday()` function. This change immediately improves the interpretability of the dataset.

When `label=TRUE` is activated, the default behavior of the function is to return the abbreviated, three-letter weekday names (e.g., Sun, Mon, Sat). This format strikes an optimal balance between clarity and conciseness, making it particularly useful for appending weekday names to large data frame columns or for use as axis labels where space is limited. Crucially, when character labels are returned, lubridate automatically converts the output column into an ordered factor. This factor structure is highly beneficial in R for ensuring that visualizations and grouping operations maintain the correct chronological order of days (Monday followed by Tuesday, etc.), rather than resorting to simple alphabetical sorting.

It is important to note that even when using character labels (`label=TRUE`), the optional `week_start` argument remains relevant. While it no longer affects the name of the day displayed, it dictates the internal ordering of the resulting factor. For instance, if you require the factor levels to begin with Monday for visualization purposes, you would use both `label=TRUE` and `week_start=1`. The following example demonstrates the basic application of abbreviated labels:

#find day of week
df$weekday <- wday(df$date, label=TRUE)

#view updated data frame
df

date sales weekday
1 2020-10-11 435 Sun
2 2020-10-19 768 Mon
3 2020-10-31 945 Sat

Method 4: Generating Full Day Names

For scenarios demanding the highest level of detail and readability, such as final executive summaries or public-facing dashboards, displaying the full name of the day (e.g., Sunday, Monday, Tuesday) is necessary. This format eliminates any potential ambiguity arising from abbreviations and offers the most straightforward interpretation for any audience. Achieving this full textual representation requires a slight extension of the previous method, combining the `label=TRUE` argument with the `abbr=FALSE` argument.

The `abbr=FALSE` parameter serves to override the default abbreviation behavior inherent to `label=TRUE`. By setting both arguments, we explicitly instruct the `wday()` function to return the

complete, descriptive name for each day. Although this approach consumes more space within a data frame or output table, the gain in clarity often justifies the increased length.

As with Method 3, the resulting column type remains an ordered factor, preserving the chronological sequence of the days. This feature ensures that even with the full names displayed, standard R operations like plotting or statistical grouping will correctly interpret the temporal ordering. This meticulous control over output formatting is a hallmark of the lubridate package, allowing analysts to transition smoothly from calculation to presentation without complex post-processing steps. The execution below demonstrates this final output format:

#find day of week
df$weekday <- wday(df$date, label=TRUE, abbr=FALSE)

#view updated data frame
df

date sales weekday
1 2020-10-11 435 Sunday
2 2020-10-19 768 Monday
3 2020-10-31 945 Saturday

Summary of Key Parameters for `wday()`

The power of the `wday()` function lies in its simple yet effective parameter system, which provides comprehensive control over the output format. Successfully utilizing this function requires a clear understanding of how the three main arguments interact to produce either numeric indices or factor labels, respecting different cultural and technical standards.

Choosing the appropriate combination of these parameters is essential for ensuring that the temporal data extracted is not only accurate but also formatted perfectly for its intended use--be it backend processing, statistical modeling, or direct presentation. The following summary clarifies the effect of each key argument and its primary use case:

No arguments (Default): This is the simplest call, yielding the numeric index (1-7), where the week begins on Sunday (Sunday=1). This is standard for internal calculations in systems that adhere to the US week definition.
`week_start=1`: This argument is used to shift the beginning of the week to Monday, assigning Monday=1 in the numeric output. It is crucial for compliance with the ISO 8601 international standard. When combined with **`label=TRUE`**, it controls the correct chronological ordering of the resulting factor levels.
`label=TRUE`: This setting transforms the output from a numeric integer to an ordered factor

composed of character labels. By default, these labels are abbreviated (e.g., Tue, Wed), maximizing efficiency in display.
`abbr=FALSE`: Used exclusively in conjunction with `label=TRUE`, this argument compels the function to return the full, unabbreviated name of the weekday (e.g., Tuesday, Wednesday).

By mastering these straightforward parameter adjustments, data analysts gain the ability to perform rapid and highly effective date manipulation using the robust lubridate package within the R programming environment. This functionality is a cornerstone of professional data processing workflows.

Note: Comprehensive and detailed documentation for the entire lubridate package, including advanced use cases for the `wday()` function, can always be found on the official CRAN documentation site for reliable, authoritative reference.

Additional Resources

For those interested in exploring more advanced techniques involving time series analysis, complex date arithmetic, or alternative date formatting methods in R, the following supplementary resources offer further guidance and practical examples. These tutorials cover other common operations necessary for robust temporal data handling: