How to Easily Convert Dates to Numeric Values in R

Authored by

stats writer

December 3, 2025

Introduction: Why Convert Dates to Numeric Values in R?

Handling date and time data efficiently is a fundamental requirement in data analysis, especially when working within the **R programming environment**. While R offers specialized classes like Date and **POSIXct** for temporal data, converting these objects into numerical representations is often essential for specific analytical tasks, such as time series modeling, machine learning feature engineering, or calculating precise time differences. Standard **R functions** designed for type coercion--including **as.numeric()**, **as.double()**, and **as.integer()**--provide built-in mechanisms for this transformation.

The core principle behind converting a date object to a numeric one in R involves calculating the elapsed time relative to a fixed starting point known as the **Epoch**. For R, and many other computing systems, this reference point is typically January 1, 1970, 00:00:00 UTC. When we apply a function like **as.numeric()** to a date object, the output is a single numerical value representing the total count of time units (usually seconds or days) that have elapsed since this initial date.

This guide explores two primary and highly effective methodologies for achieving date-to-numeric conversion in R. The first method utilizes the powerful, built-in base R coercion functions, which yield a standardized numeric offset. The second method leverages the specialized capabilities of the **lubridate package**, allowing for the extraction of individual numeric components (year, month, day, hour, etc.) from the date object, providing flexibility depending on the analytical requirement.

Method 1: Coercing Dates Using Base R's as.numeric()

There are two methods you can use to convert date values to numeric values in R:

Method 1: Use as.numeric()

as.numeric(my_date)

This will return the number of seconds that have passed between your date object and 1/1/1970.

The **as.numeric()** function is the most straightforward mechanism in base R for converting temporal objects into their underlying numerical representation. When applied to a standard **Date** class object, **as.numeric()** returns the count of days since the **Epoch** (1970-01-01). However, when applied to a **POSIXct** or **POSIXlt** object, which includes time components, it returns the number of **seconds** that have elapsed since the Epoch. Understanding which time unit is returned is critical for correct interpretation of the resulting numeric value.

This method provides a continuous numerical scale for time, making it exceptionally valuable for

mathematical operations. For instance, subtracting the numeric value of one date from another immediately yields the difference in seconds, which can then be easily scaled to minutes, hours, or days. This standardization into a measurable unit since 1970 ensures data consistency across various datasets and computational environments. The resulting numeric data is highly suitable for statistical algorithms that require continuous, ordered input.

Method 2: Component Extraction using the lubridate Package

Method 2: Use Functions from the lubridate package

library(lubridate)

```
#get seconds value in date object
second(my_date)

#get minutes value in date object
minute(my_date)

...
#get year value in date object
year(my_date)
```

This will return the value for the seconds, minutes, years, etc. from your date object.

While **as.numeric()** provides a single, unified numeric timestamp, data analysis often requires extracting specific components of a date--such as the month number, the day of the week, or the year--to use as categorical or independent variables. The **lubridate package**, part of the **Tidyverse** ecosystem, is specifically designed to simplify working with dates and times in R. It offers a highly intuitive suite of functions for extracting these individual components directly as numeric values.

By loading **lubridate**, you gain access to functions like **year()**, **month()**, **day()**, **hour()**, **minute()**, and **second()**. Unlike **as.numeric()**, which calculates the offset from the **Epoch**, these functions isolate and return the specific numerical value associated with that component within the date string. For example, if your date is "2023-10-26 14:30:00", calling **year()** will return 2023, and calling **hour()** will return 14.

This approach is particularly valuable for feature engineering in predictive modeling. For instance, if you suspect that sales performance is seasonal, extracting the numeric month (1 through 12) allows you to treat seasonality as a numerical predictor in your models. The use of **lubridate** greatly enhances code readability and reduces the complexity associated with manual string

manipulation or date formatting necessary in other systems, allowing analysts to quickly prepare temporal features.

Practical Implementation: Using as.numeric() for Time Scaling

The following examples show how to use each method in practice.

Method 1: Use as.numeric()

The following code shows how to convert a date object to numeric using the **as.numeric()** function:

```
#create date object
my_date <- as.POSIXct("10/14/2021 5:35:00 PM", format="%m/%d/%Y %H:%M:%S %p")

#view date object
my_date

"2021-10-14 05:35:00 UTC"

#convert date object to number of seconds since 1/1/1970
as.numeric(my_date)

1634189700

#convert date object to number of days since 1/1/1970
as.numeric(my_date) / 86400

18914.23

#convert date object to number of years since 1/1/1970
as.numeric(my_date) / 86400 / 365

51.81982
```

In the example above, we first define our temporal variable, **my_date**, utilizing the **as.POSIXct** function. This function ensures that the date and time string is parsed correctly and stored internally as a precise timestamp, which is essential for accurate conversions involving seconds. The output confirms that R recognizes the object as being in **UTC** (Coordinated Universal Time), which is the default for high-resolution timestamps.

When **as.numeric(my_date)** is executed, the result is 1,634,189,700. This massive integer represents the total number of seconds elapsed between the standard **Epoch** start date (January

1, 1970) and the specific timestamp stored in **my_date** (October 14, 2021, 5:35:00 PM UTC). This conversion provides a highly standardized, continuous numeric metric of time that is machine-readable.

To transform this result into more human-readable units, simple arithmetic scaling is applied. Since there are 86,400 seconds in a day (24 hours * 60 minutes * 60 seconds), dividing the total seconds by 86,400 yields the number of days since the Epoch (18,914.23 days). Further division by 365 (or 365.25 for higher accuracy in large spans) converts this into years (51.81982 years). This demonstrates the flexibility of using the base **as.numeric()** output for various time calculations, enabling straightforward comparisons across decades.

Interpreting the Standardized Numeric Output

Based on the output we can see:

There is a difference of **1,634,189,700 seconds** between our date object and 1/1/1970.
There is a difference of **18,914.23 days** between our date object and 1/1/1970.
There is a difference of **51.81982 years** between our date object and 1/1/1970.

The interpretation of the numeric output derived from **as.numeric()** depends entirely on the initial R class of the temporal object. For **Date** objects (which lack a time component), the resulting integer represents whole days. For high-resolution time objects like **POSIXct**, the result is in seconds, providing maximum granularity for precise calculations. Analysts must confirm the original object class to prevent misinterpreting seconds as days, or vice versa.

Understanding the concept of the time offset is crucial for utilizing this method effectively in advanced statistical modeling. A larger numeric value simply means the date is further removed in time from January 1, 1970. This continuous numerical scale allows time to be treated as a quantitative variable in regression models or clustering algorithms, where distance metrics are key. Furthermore, the standardization provided by the **R language** ensures that time zone considerations are handled consistently, particularly when using **POSIXct**, which defaults to or utilizes UTC unless explicit time zone arguments are provided during creation.

It is important to remember that the precision of the resulting numeric value (seconds, days) dictates the potential error in subsequent calculations. If sub-second precision is not required, converting to days or even years (as shown in the code example) can simplify analysis and improve computational performance, although the loss of granular timing information should be factored into the methodology before aggregation.

Practical Implementation: Component Extraction using lubridate

Method 2: Use Functions from the lubridate Package

The following code shows how to convert a date object to numeric using functions from the package in R:

```
library(lubridate)

#create date object
my_date <- as.POSIXct("10/14/2021 5:35:00 PM", format="%m/%d/%Y %H:%M:%S %p")

#view date object
my_date

"2021-10-14 05:35:00 UTC"

#extract various numerical values from date object
second(my_date)

0

minute(my_date)

35

hour(my_date)

5

day(my_date)

14

month(my_date)

10

year(my_date)

2021
```

The application of **lubridate** begins with loading the package using the **library()** function. We reuse the same **my_date** object, created using **as.POSIXct**, ensuring it is a properly formatted

date-time object that **lubridate** functions can parse efficiently. The power of this package lies in its ability to strip away the time structure and return only the numeric attribute of interest, without relying on the Epoch calculation.

For example, executing **month(my_date)** returns the integer 10, corresponding exactly to October, the tenth month of the year. Similarly, **hour(my_date)** returns 5, which represents the hour component (5 AM in this example, based on the internal UTC conversion). It is critical to note that the output of **hour()**, **minute()**, and **second()** will reflect the internal **POSIXct** representation, which is often UTC, regardless of the input time zone unless explicitly handled.

Using these functions, we can extract the seconds, minutes, hours, days, months, and year values from our date object. This methodology is preferred when the relative timing structure (e.g., "is it morning?") is more important than the absolute time duration since the **Epoch**. For analysts focusing on periodicity, seasonality, or daily trends, extracting components provides ready-to-use numeric features for immediate modeling without complex division or scaling required by the **as.numeric()** approach.

Comparing Conversion Methods: When to Use Which?

Choosing between the base R **as.numeric()** method and the **lubridate** component extraction method depends entirely on the goal of the data manipulation. If the objective is to calculate the time distance between two events, perform high-precision time series comparisons, or integrate time as a continuous variable into a statistical model, the **as.numeric()** method is superior. It provides a standardized, continuous metric--typically seconds since 1970--which is ideal for mathematical operations and continuity checks.

Conversely, if the primary goal is to isolate specific periodic features or analyze how behavior changes based on calendar components (e.g., comparing productivity by month or analyzing traffic patterns by hour of the day), the **lubridate package** offers a cleaner, more direct solution. Its functions return intuitive integers (1-12 for month, 0-23 for hour), making data preparation for categorical analysis straightforward, without needing to worry about the underlying seconds calculation.

In essence, **as.numeric()** focuses on the chronological distance from the Epoch, while **lubridate** focuses on the intrinsic numerical properties of the calendar date itself. Experienced **R users** often combine both methods: using **as.numeric()** for time difference calculations and utilizing **lubridate** functions for feature engineering based on seasonality or cyclical patterns, achieving maximum analytical flexibility.

Conclusion: Mastering Date Conversion in R

Converting date objects into numerical formats is a crucial skill for advanced data manipulation in R. Whether you require a continuous numeric representation for distance calculations or discrete numeric components for feature engineering, R provides robust tools. Base R offers functions like **as.numeric()** to convert dates into the number of seconds or days since the 1970 Epoch, providing a scalable numeric timestamp suitable for continuous modeling.

For tasks demanding greater readability and the isolation of specific temporal attributes, the specialized functions within the **lubridate package** offer unparalleled ease of use. By mastering both the coercion method (**as.numeric()**) and the component extraction method (**lubridate** functions), analysts can confidently prepare time-series data for sophisticated modeling and analysis, ensuring that temporal variables are correctly handled and interpreted within the chosen statistical framework.

The following tutorials explain how to perform other common conversions in R: