How to Aggregate Daily Data to Monthly and Yearly in R?

Authored by

stats writer

December 20, 2025

The process of transforming high-frequency data, such as daily records, into lower-frequency summaries (monthly, quarterly, or yearly) is a fundamental task in time series analysis and business intelligence. While older methods in R might rely solely on the base function `aggregate()`, modern data science workflows leverage the power and efficiency offered by the dplyr package, often in conjunction with the specialized date-handling capabilities of the lubridate package.

This tutorial provides an expert-level guide on generating clean, period-based aggregates from daily data. Proper data aggregation is crucial for smoothing out daily noise, identifying long-term trends, and calculating relevant summary statistic metrics such as means, sums, or medians over defined intervals. We will demonstrate how to systematically group daily observations into specific time buckets--be they weekly, monthly, or yearly--using the streamlined Tidyverse approach.

The Necessity of Time Series Aggregation

Analyzing raw daily data can often be overwhelming due to high volatility and volume. By aggregating daily figures--such as sales volumes, temperature readings, or website traffic--into larger, consistent units like weeks or months, analysts can achieve a clearer perspective on underlying patterns and seasonal cycles. This transformation simplifies visual representations and supports more robust statistical modeling, which is essential for accurate forecasting and strategic planning. The ability to perform this grouping efficiently is paramount in a high-performance environment like R.

In the context of data aggregation, we must define clear boundaries for our new time intervals. For instance, when aggregating daily sales to monthly sales, every daily observation must be correctly assigned to its respective calendar month. This process requires precise date manipulation, which is where the dedicated functions provided by the Tidyverse ecosystem excel, offering superior readability and performance compared to base R functions for complex grouping operations.

Prerequisites: Essential R Packages

To successfully execute time-based data aggregation, we rely on two pivotal packages within the Tidyverse. First, the lubridate package, developed by Hadley Wickham, simplifies working with dates and times by providing intuitive functions for parsing, manipulating, and rounding date objects. This package is critical for establishing the time periods (e.g., the first day of the week or month) that define our groups.

Second, the dplyr package provides a consistent grammar for data manipulation. Its core functions--specifically `group_by()` and `summarize()`--enable us to execute split-apply-combine strategies with great efficiency. Once lubridate package creates the aggregation variable (e.g., the

month start date), `group_by()` organizes the daily records based on this new variable, and `summarize()` then calculates the desired summary statistic (like the average or total sales) for each defined group.

Preparing the Daily Dataset for Analysis

Before any aggregation can take place, we must ensure we have a well-formed data structure containing daily observations and a proper date variable. For demonstration purposes, we will create a synthetic data frame that tracks daily sales over a 100-day period. This example uses the `set.seed()` function to ensure that the random data generated is reproducible, allowing users to replicate the exact results shown in this tutorial.

The data frame `df` contains two columns: `date`, which uses the official R date format (created by adding integers to an initial date string), and `sales`, which is populated by 100 random numbers generated between 20 and 50 using `runif()`. Reviewing the initial rows confirms the structure of our daily time series data, which is now ready for transformation.

#make this example reproducible
set.seed(1)

#create data frame
df <- data.frame(date = as.Date("2020-12-01") + 0:99,
sales = runif(100, 20, 50))

#view first six rows
head(df)

date sales
1 2020-12-01 27.96526
2 2020-12-02 31.16372
3 2020-12-03 37.18560
4 2020-12-04 47.24623
5 2020-12-05 26.05046
6 2020-12-06 46.95169

Understanding floor_date() for Time Grouping

To successfully perform data aggregation by time period, we must first assign a standardized timestamp to every observation that represents the start of its respective aggregation window. The `floor_date()` function from the lubridate package is designed precisely for this purpose. It effectively "rounds down" a date object to the start of a specified time unit, thereby creating a

common key for grouping. For example, applying `floor_date()` with the unit "month" to any date in December 2020 will return `2020-12-01`, which acts as the monthly identifier.

The syntax for `floor_date()` is straightforward yet powerful, offering flexibility across various time granularities. Understanding its two main arguments is key to effective date manipulation in R. This function provides the essential bridge between raw daily data and the structured groups required by the dplyr package's summarizing capabilities.

floor_date(x, unit)

The arguments define the scope of the rounding operation, determining how the daily observations are categorized:

x: This argument requires a vector containing time series data, typically date or datetime objects, which need to be rounded.
unit: This critical argument specifies the desired time unit for rounding. Available options are extensive and include `second`, `minute`, `hour`, `day`, `week`, `month`, `bimonth`, `quarter`, `halfyear`, and `year`. Choosing the correct unit determines the level of aggregation.

Aggregating to Weekly Averages

Aggregating daily sales data to weekly averages involves three distinct steps using the Tidyverse pipe (`%>%`) structure: defining the weekly grouping variable, grouping the data based on this new variable, and calculating the mean sales for each group. First, we apply `floor_date(df$date, "week")`, which truncates each date down to the beginning of its respective week. This result is stored in a new column, `df$week`.

Next, we use the dplyr package functions. The `group_by(week)` command tells R to treat all observations sharing the same week start date as a single unit. Finally, `summarize(mean = mean(sales))` calculates the arithmetic mean of the `sales` column within each of those defined weekly groups. This method is highly efficient for calculating a summary statistic over a custom time period.

The resulting output is a tidy data frame (a tibble) showing the calculated average sales for each week represented in the original dataset. Note how the first week, `2020-11-29`, captures the days leading up to the first full week of data starting December 1st, demonstrating the precise period-start logic employed by the lubridate package.

library(lubridate)
library(dplyr)

```
#round dates down to week
df$week <- floor_date(df$date, "week")

#find mean sales by week
df %>%
group_by(week) %>%
summarize(mean = mean(sales))
```

```
# A tibble: 15 x 2
week mean

1 2020-11-29 33.9
2 2020-12-06 35.3
3 2020-12-13 39.0
4 2020-12-20 34.4
5 2020-12-27 33.6
6 2021-01-03 35.9
7 2021-01-10 37.8
8 2021-01-17 36.8
9 2021-01-24 32.8
10 2021-01-31 33.9
11 2021-02-07 34.1
12 2021-02-14 41.6
13 2021-02-21 31.8
14 2021-02-28 35.2
15 2021-03-07 37.1
```


Aggregating to Monthly Averages

Shifting the aggregation granularity from weekly to monthly only requires a minor adjustment in the `floor_date()` function. Instead of specifying `"week"` as the unit, we use `"month"`. This modification instructs the lubridate package to assign the first day of the calendar month (e.g., `2021-01-01`) as the group identifier for all daily observations falling within that month. The subsequent steps involving `group_by()` and `summarize()` remain identical, demonstrating the consistent methodology of the Tidyverse framework for time-based data aggregation.

The resulting output clearly shows the average sales calculated across the full duration of December 2020, January 2021, and February 2021, as well as the partial data available for March 2021. This simplification drastically reduces the number of rows while providing insightful, high-level context on sales performance over time, essential for trend analysis in any time series

dataset.

library(lubridate)
library(dplyr)

#round dates down to week
df$month <- floor_date(df$date, "month")

#find mean sales by month
df %>%
group_by(month) %>%
summarize(mean = mean(sales))

A tibble: 4 x 2
month mean

1 2020-12-01 35.3
2 2021-01-01 35.6
3 2021-02-01 35.2
4 2021-03-01 37.0

Aggregating to Yearly Averages

For the broadest view of the data, we aggregate the daily sales into yearly averages. This process further simplifies the dataset, especially when dealing with data spanning multiple years. Similar to the monthly aggregation, this transformation is achieved by setting the `unit` argument in `floor_date()` to `"year"`. This ensures that every date is rounded down to January 1st of its corresponding year, establishing the yearly group identifier.

Given that our sample data spans parts of 2020 and 2021, the yearly aggregation provides two overall averages. While the 2020 average is based only on December data, and the 2021 average is based on January through early March, this operation successfully condenses the 100 daily entries into just two representative rows. This illustrates the power of `floor_date()` and the dplyr package pipeline for high-level longitudinal analysis.

library(lubridate)
library(dplyr)

#round dates down to week
df$year <- floor_date(df$date, "year")

#find mean sales by month

```
df %>%
group_by(year) %>%
summarize(mean = mean(sales))

# A tibble: 2 x 2
year mean

1 2020-01-01 35.3
2 2021-01-01 35.7
```

Conclusion: Beyond the Mean in Summary Statistics

Throughout these examples, we consistently used the `mean()` function within the `summarize()` command to calculate the average sales for each time period. However, the flexibility of the dplyr package allows analysts to use any valid summary statistic function supported by R. Depending on the nature of the data and the analytical goals, alternative metrics may be more appropriate for describing the aggregated time series data.

For instance, if the data is highly skewed or contains influential outliers, the `median()` might offer a more robust representation of the central tendency than the mean. If the goal is to track accumulated volume (like total rainfall or total visitors), the `sum()` function should be used. Other useful functions include `max()`, `min()`, `sd()` (standard deviation), or even custom functions defined by the user. The power of this aggregation methodology lies in its standardized approach: regardless of the chosen time unit or statistical function, the core Tidyverse pipeline using lubridate package and dplyr package remains efficient and syntactically clean.

By mastering the integration of `floor_date()`, `group_by()`, and `summarize()`, practitioners gain a powerful toolkit for transforming high-frequency data into meaningful aggregated insights, simplifying complex data analysis tasks significantly.

How to Calculate the Mean by Group in R
How to Calculate Cumulative Sums in R
How to Plot a Time Series in R