

How to Create a Time Series in R

Authored by
stats writer

November 19, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Create a Time Series in R*. PSYCHOLOGICAL SCALES.
Retrieved from <https://scales.arabpsychology.com/?p=97092>

Analyzing time series data is a fundamental skill in fields ranging from finance and economics to meteorology and engineering. In the R programming language, this complex data structure is handled efficiently using specialized functions. The core concept involves converting a simple vector of sequential measurements into a structured time series object, which unlocks powerful analytical tools. This transformation is primarily achieved using the crucial **ts()** function.

The time series object in R is not merely a list of numbers; it's an enriched data structure that incorporates temporal attributes like start time, end time, and observation frequency. By properly defining these attributes, R can automatically handle indexing, plotting, and decomposition of the data. Understanding how to correctly initialize this object is the first and most critical step toward performing robust time series forecasting and analysis.

Once the data is structured, R provides immediate access to visualization and decomposition methods. For instance, the basic **plot()** function can generate insightful visualizations immediately, while advanced functions like `decompose()` allow analysts to separate the data into its constituent parts: the underlying trend, the predictable seasonality, and the residual random component. This article will guide you through the process of creating, inspecting, and visualizing a valid time series object in R.

The Essential ts() Function Syntax

The simplest and most effective mechanism for generating a time series object in R relies on the **ts()** function. This function takes a vector or matrix of observed values and overlays the necessary temporal metadata. Without defining these parameters correctly, R treats the data as a simple numerical sequence rather than a chronologically ordered series.

To successfully utilize this function, you must provide the raw data along with parameters defining when the series begins and the rate at which observations occur. This configuration ensures that R interprets the data sequence accurately, mapping each value to its correct point in time. Below is the fundamental syntax structure required for the **ts()** function:

```
ts(data, start, end, frequency)
```

The arguments define the context of the data:

data: This is the primary input, representing a vector or matrix containing the observed time series values.

start: Specifies the time point corresponding to the first observation in the **data** vector. This can be a single year (for yearly data) or a vector (e.g., `c(Year, Period)` for sub-yearly data).

end: Defines the time point corresponding to the last observation. Although optional, it helps confirm the expected duration.

frequency: This is a critical argument that determines the number of observations recorded per unit of time (e.g., 1 for yearly, 4 for quarterly, or 12 for monthly data). This setting is essential for identifying seasonality.

The following examples show how to use this function to create different time series objects in practice.

Practical Example: Modeling Monthly Data

When dealing with data collected on a sub-yearly basis, such as monthly or quarterly figures, it is essential to set the `frequency` parameter correctly. For monthly data, the `frequency` must be set to 12. This tells R that there are 12 observations within a single yearly cycle, allowing the system to correctly identify and visualize seasonal patterns.

Consider a scenario where a retail store recorded sales figures over 20 consecutive months, commencing in October 2023. We first define the raw sales data as a vector named `data`. This vector holds the sequence of sales values collected over the period.

#create vector of 20 values

```
data <- c(6, 7, 7, 7, 8, 5, 8, 9, 4, 9, 12, 14, 14, 15, 18, 24, 20, 15, 24, 26)
```

To convert this vector into a proper time series object, we call the `ts()` function. We specify the start time as October (period 10) of 2023, using the format `c(2023, 10)`, and set the `frequency` to 12, reflecting the monthly observation rate. The resulting object, `ts_data`, is now structured chronologically.

#create time series object from vector

```
ts_data <- ts(data, start=c(2023, 10), frequency=12)
```

#view time series object

```
ts_data
```

```
Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2023 6 7 7
2024 7 8 5 8 9 4 9 12 14 14 15 18
2025 24 20 15 24 26
```

Upon inspecting the output, notice how R has automatically indexed the sales values, associating them with the correct months from October 2023 through May 2025. This automatic dating and indexing is the key advantage of using the time series object structure over a simple numerical vector. We can now proceed to confirm the structure.

Verifying Time Series Object Properties

After creating the time series object, it is good practice to formally verify its structure and class. This confirms that the `ts()` function executed successfully and that R recognizes the data as a specialized time series structure rather than a generic numerical object. This is achieved using the standard `class()` function.

By querying the class of the newly created variable `ts_data`, R confirms that it belongs to the "ts" class, which stands for time series. This verification step is simple but ensures compatibility with all subsequent time series analytical functions built into R packages.

```
#display class of ts_data object
```

```
class(ts_data)
```

```
"ts"
```

Practical Example: Working with Yearly Data

When dealing with data collected annually, the configuration becomes much simpler. Since there is only one observation per time unit (the year), the `frequency` parameter should be set to 1. This is often the default setting if omitted, but explicitly setting `frequency=1` is recommended practice for clarity and robustness.

Let's reuse the same 20 sales figures, but now assume they represent annual sales totals collected over 20 consecutive years, beginning in the year 2000. We start by defining the same vector of sales values:

```
#create vector of 20 values
```

```
data <- c(6, 7, 7, 7, 8, 5, 8, 9, 4, 9, 12, 14, 14, 15, 18, 24, 20, 15, 24, 26)
```

We then create the time series object, setting the starting year to 2000 and the `frequency` to 1. Note that the `start` parameter only requires a single integer (the year) when `frequency=1`.

```
#create time series object from vector
```

```
ts_data <- ts(data, start=2000, frequency=1)
```

```
#view time series object
```

```
Time Series:
```

```
Start = 2000
```

```
End = 2019
```

```
Frequency = 1
```

6 7 7 7 8 5 8 9 4 9 12 14 14 15 18 24 20 15 24 26

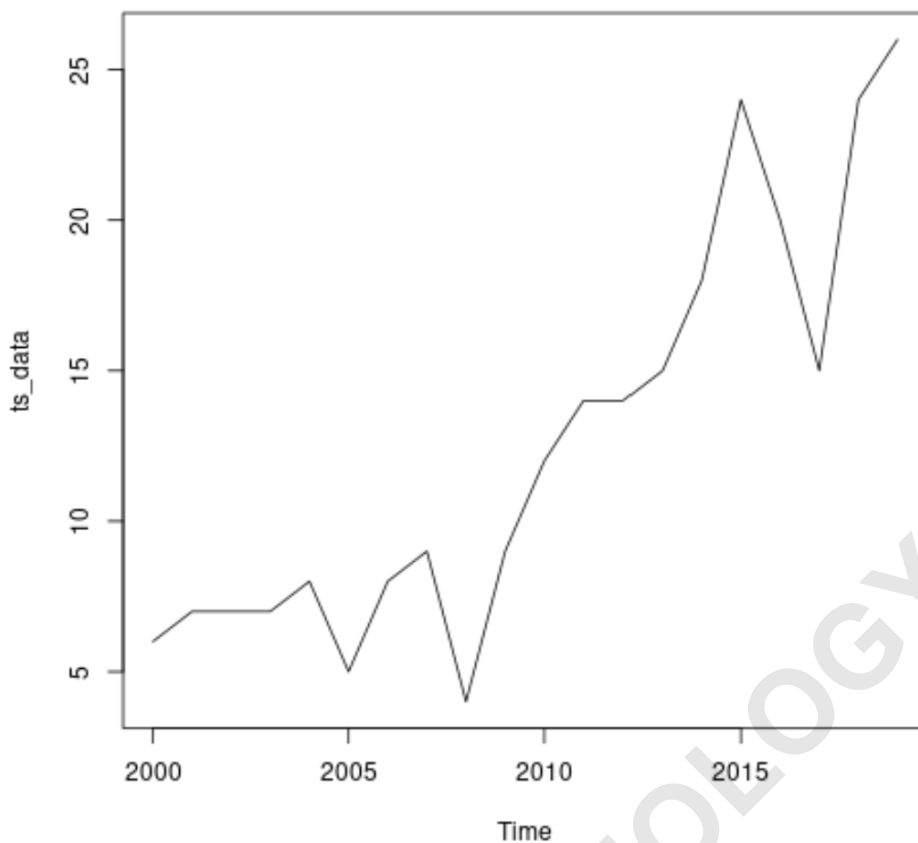
The resulting object clearly displays its metadata: the series spans from 2000 to 2019, confirming that R has correctly mapped the 20 sales values to their respective years. This yearly object is now ready for visualization and analysis.

Visualizing Time Series with `plot()`

One of the immediate benefits of converting data into an R time series object is the ease of visualization. The generic `plot()` function is context-aware; when passed a time series object, it automatically generates a line plot where the x-axis represents time and the y-axis represents the observed values. This provides a crucial visual overview of time series behavior, helping identify trends, cycles, and volatility.

Using the yearly sales data created previously, we can generate a basic plot with a single command. This quick visualization allows us to immediately observe how sales have evolved over the 20-year period.

```
#create line plot of time series data  
plot(ts_data)
```



As evident in the output image, the horizontal axis is automatically labeled with the years (2000 to 2019), and the vertical axis displays the sales counts. While this default plot is useful, professional analysis often requires clearer labeling and styling for presentations or reports.

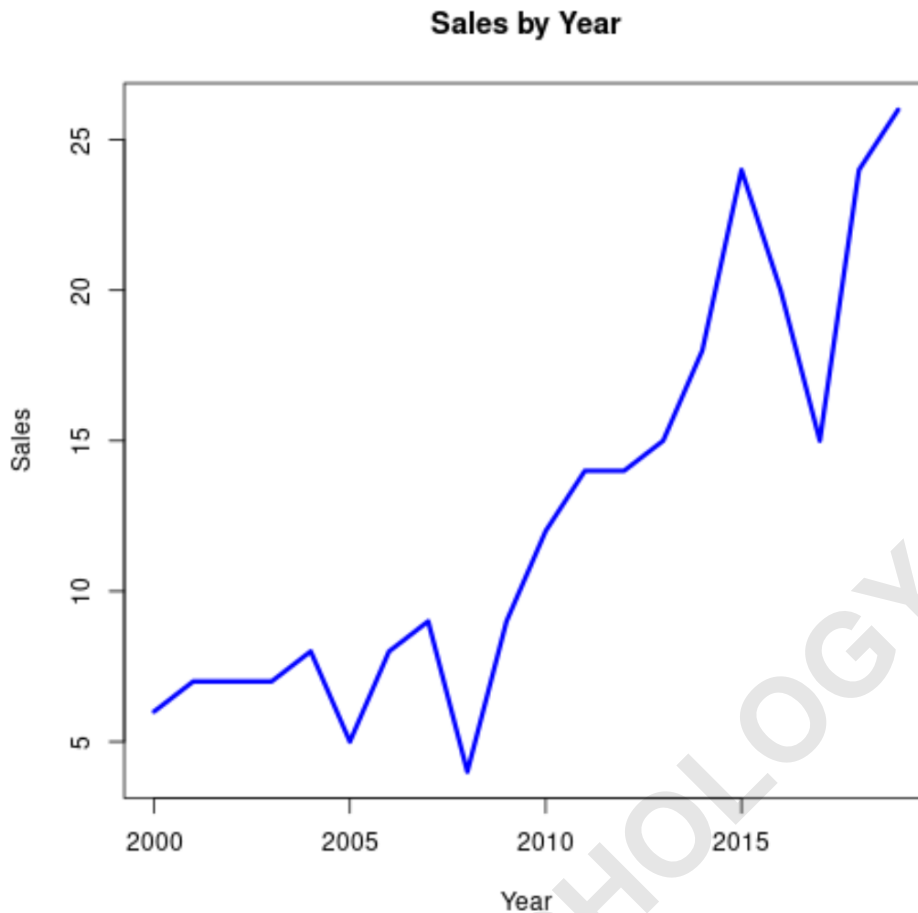
Customizing Time Series Plots

The **plot()** function in R is highly customizable, allowing users to enhance readability and aesthetic appeal. By passing additional graphical parameters, you can control elements such as axis labels, titles, line color, and line thickness. This level of customization ensures that the visual representation accurately reflects the insights derived from the data.

Key parameters for customization include `xlab` (x-axis label), `ylab` (y-axis label), `main` (plot title), `col` (line color), and `lwd` (line width). Applying these parameters transforms the basic plot into a more polished and informative graphic, improving communication of the analysis results.

#create line plot with custom x-axis, y-axis, title, line color and line width

```
plot(ts_data, xlab='Year', ylab='Sales', main='Sales by Year', col='blue', lwd=3)
```



Feel free to modify the arguments in the **plot()** function to create the exact time series plot you need, ensuring your visualizations are both accurate and compelling.

Conclusion: Next Steps in Time Series Analysis

Successfully generating a time series object using **ts()** is the foundational step for any temporal analysis in R. Once the data is correctly structured and visualized, analysts can move on to more advanced techniques like modeling, forecasting, and decomposition.

A standard next step, particularly for data exhibiting strong seasonality (like the monthly data example), is to use the `decompose()` function. This function automatically separates the observed data into three distinct components: the underlying long-term trend (T), the seasonal variation (S), and the irregular or random remainder (R). Understanding these components is vital for accurate forecasting and identifying the true drivers of variation within the series. Mastering the creation of the time series object is the key that unlocks these powerful analytical capabilities.