Forecasting Time Series Data with the forecast() Function in R: A Step-by-Step Guide

Authored by

Mohammed looti

November 12, 2025

In the realm of modern data science, the analysis of sequential observations--or **time series** data--is fundamentally tied to the ability to project future outcomes. This predictive capability is a core requirement across diverse sectors, including quantitative finance, inventory management, and macroeconomic planning. Accurate **time series** forecasting enables organizations to mitigate risk and capitalize on anticipated trends, making it an indispensable skill for any analyst working in **R**.

The **R** environment is uniquely equipped for these complex statistical tasks, largely due to specialized packages designed for ease of use and robustness. Central to this process is the powerful **forecast package**, which streamlines the generation of reliable future estimates. Within this ecosystem, the primary tool for automated prediction is the **forecast()** function. This function abstracts the complexity of fitting sophisticated models like **ARIMA** or **Exponential Smoothing (ETS)**, providing a simple interface to obtain statistically sound projections.

Mastering the **forecast()** function is essential for efficient analysis. This article provides a comprehensive guide to its syntax, data prerequisites, and interpretation of the resulting prediction intervals.

Defining the Parameters of the forecast() Function

To maximize the utility of the **forecast()** function, it is crucial to understand its syntax and the role of each argument. The function is designed to be highly versatile, capable of accepting various statistical model outputs or raw time series data as input, and allowing precise control over the prediction horizon and the certainty of the estimates.

The fundamental structure for invoking the function is defined as follows, requiring only a few key parameters to execute a complete forecast:

forecast(object, h, level, ...)

The primary arguments that govern the prediction outcome are detailed below:

object: This is the mandatory input, which can be either a fitted statistical model (such as an **ARIMA** model, or one derived from the **forecast package**) or the raw **time series** object itself.
h: This parameter specifies the forecast horizon, dictating the exact **number of periods** into the future for which the prediction should be calculated. Setting `h=12`, for instance, projects the data one year forward if the data frequency is monthly.
level: This optional but highly useful argument defines the required **confidence level** for the output **prediction intervals**, expressed as a decimal value (e.g., 0.90 for 90% confidence).

It is important to note the default behavior of the function: if the `level` argument is omitted, the **forecast()** function automatically calculates and returns prediction intervals at both the 80% and

95% **confidence level**s. Analysts should also remember the fundamental trade-off in statistical inference: increasing the specified `level` inherently widens the resulting prediction interval, reflecting the increased certainty required to encompass the true, unknown future value.

Structuring Input Data: The R Time Series Object

A prerequisite for accurate forecasting in **R** is correctly structuring the raw data into a recognizable **time series** object. Without this essential transformation, specialized statistical functions, including **forecast()**, cannot correctly interpret temporal dependencies or cyclical patterns. This process is handled by R's built-in `ts()` function, which takes a simple vector of observations and endows it with temporal metadata.

The `ts()` function requires the user to define two crucial parameters beyond the data vector itself: the `start` period and the `frequency`. The `start` parameter establishes the chronological index of the first observation, typically specified as a year and month/quarter (e.g., `c(2023, 10)`). The `frequency` parameter is vital for capturing seasonality, indicating the number of observations within a typical cycle (e.g., `frequency=12` for monthly data or `frequency=4` for quarterly data).

Correctly defining these parameters ensures that subsequent forecasting algorithms, such as those automatically deployed by **forecast()**, properly account for recurring seasonal effects. If, for instance, we are analyzing monthly sales data, setting `frequency=12` signals that patterns should be searched for and modeled over a full twelve-month span, leading to significantly more reliable predictions. This foundational step is non-negotiable for producing high-quality forecasts.

Practical Demonstration: Generating a Forecast in R

To solidify our understanding, we will walk through a practical example involving a sample dataset representing monthly observations. Our aim is to simulate a scenario where data points are collected sequentially, spanning from October 2023 through May 2025. This structured historical data will serve as the input for our predictive model.

First, we define the sequence of observations as a simple vector. We then convert this vector into a time series object named `ts_data` using the `ts()` function. For this example, we set the starting period to October 2023 (`start=c(2023, 10)`) and explicitly set the frequency to 12, confirming that the underlying model should recognize and incorporate monthly seasonality.

The following R code snippet executes these preparatory steps and displays the resulting time series object, confirming the correct chronological indexing:

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

```
#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
```

The resulting structure confirms that the data has been correctly indexed according to the specified monthly frequency, starting in October 2023. This object is now ready to be used as input for the forecasting process.

Executing the Forecast and Analyzing Default Output

With the data properly structured, we can now proceed to predict the next twelve months of observations, starting from June 2025. This process requires two main steps: loading the necessary library and calling the **forecast()** function with the specified horizon. We begin by ensuring the **forecast package** is loaded into our active **R** session using `library(forecast)`.

We then execute the function, providing `ts_data` as the input object and setting the forecast horizon `h=12`. A key feature of the **forecast package** is its ability to automatically select and fit a suitable model (often an optimized **ETS** model) when no explicit model is provided. The resulting output not only provides the single best estimate for each period but also includes the default 80% and 95% **prediction intervals**, quantifying the expected range of uncertainty.

Observe the R command and the complete output generated for the 12-period projection:

library(forecast)

```
#forecast values for 12 periods in future of time series
forecast(ts_data, h=12)
```

```
Point Forecast Lo 80 Hi 80 Lo 95 Hi 95
Jun 2025 23.74305 13.629491 33.85662 8.2756948 39.21041
Jul 2025 23.74305 12.062173 35.42393 5.8786883 41.60742
Aug 2025 23.74305 10.638219 36.84789 3.7009389 43.78517
Sep 2025 23.74305 9.313813 38.17229 1.6754342 45.81067
Oct 2025 23.74305 8.062465 39.42364 -0.2383363 47.72444
```

Nov 2025 23.74305 6.866724 40.61938 -2.0670649 49.55317
Dec 2025 23.74305 5.714367 41.77174 -3.8294430 51.31555
Jan 2026 23.74305 4.596437 42.88967 -5.5391701 53.02528
Feb 2026 23.74305 3.506129 43.97998 -7.2066512 54.69276
Mar 2026 23.74305 2.438127 45.04798 -8.8400200 56.32613
Apr 2026 23.74305 1.388172 46.09793 -10.4457875 57.93189
May 2026 23.74305 0.352787 47.13332 -12.0292721 59.51538

This output confirms that the function successfully generated 12 periods of prediction, extending the analysis from June 2025 through May 2026. The structure of the results--including the point forecasts and confidence bounds--provides a complete picture of the future trend and the associated prediction risk.

Interpreting Prediction Intervals and Forecast Uncertainty

A crucial aspect of interpreting the output from **forecast()** is understanding the distinction between the **Point Forecast** and the accompanying prediction intervals. While the point forecast represents the single most probable **point estimate** for the future value, the prediction intervals (Lo 80/Hi 80 and Lo 95/Hi 95) are far more valuable for real-world decision-making as they capture the inherent statistical uncertainty.

A prediction interval defines a specific range where we expect the actual future observation to fall, based on the statistical model. For example, the 95% interval implies that if the forecasting experiment were repeated many times, 95% of the observed future values would fall between the `Lo 95` and `Hi 95` bounds. Analyzing the results from the previous example, it is clear that the 95% interval is substantially wider than the 80% interval.

This widening is a direct consequence of increasing the required **confidence level**. To be more certain that the true outcome is captured (e.g., moving from 80% certainty to 95% certainty), the range of possibilities must expand. This statistical principle highlights why relying solely on the point estimate without considering the prediction interval can lead to significant errors in planning and risk assessment.

Customizing the Confidence Level for Specific Needs

While the default 80% and 95% intervals are standard, practical applications often demand a different degree of certainty. For specialized fields like quality control or financial risk assessment, a 90% or even 99% **confidence level** might be required. The `level` argument in the **forecast()** function provides the necessary flexibility to precisely tailor these interval bounds.

To illustrate this customization, we can modify our previous syntax to generate prediction intervals at a 90% confidence threshold. This is achieved by setting `level=0.9`. This action yields a prediction interval that offers higher confidence than the 80% default but is slightly narrower than the 95% default, providing a balanced measure of certainty for moderate-risk decisions.

The following R command demonstrates the syntax adjustment and the resulting customized output:

library(forecast)

#forecast values for 12 periods in future of time series
forecast(ts_data, h=12, level=0.9)

Point Forecast Lo 90 Hi 90
Jun 2025 23.74305 10.7624368 36.72367
Jul 2025 23.74305 8.7508056 38.73530
Aug 2025 23.74305 6.9231807 40.56293
Sep 2025 23.74305 5.2233235 42.26278
Oct 2025 23.74305 3.6172367 43.86887
Nov 2025 23.74305 2.0825193 45.40359
Dec 2025 23.74305 0.6034850 46.88262
Jan 2026 23.74305 -0.8313632 48.31747
Feb 2026 23.74305 -2.2307575 49.71686
Mar 2026 23.74305 -3.6015238 51.08763
Apr 2026 23.74305 -4.9491264 52.43523
May 2026 23.74305 -6.2780285 53.76414

The output now features the `Lo 90` and `Hi 90` columns, providing the customized lower and upper boundaries for the 90% interval. This straightforward manipulation demonstrates how the **forecast()** function is easily adaptable to meet specific business and statistical requirements, solidifying its role as an indispensable tool for predictive modeling in the R environment.

Additional Resources for R Forecasting

To further enhance your proficiency in statistical computing and forecasting within the R environment, explore the following resources and tutorials that cover common analytical tasks:

Featured Posts

[5 Statistical Biases to Avoid]

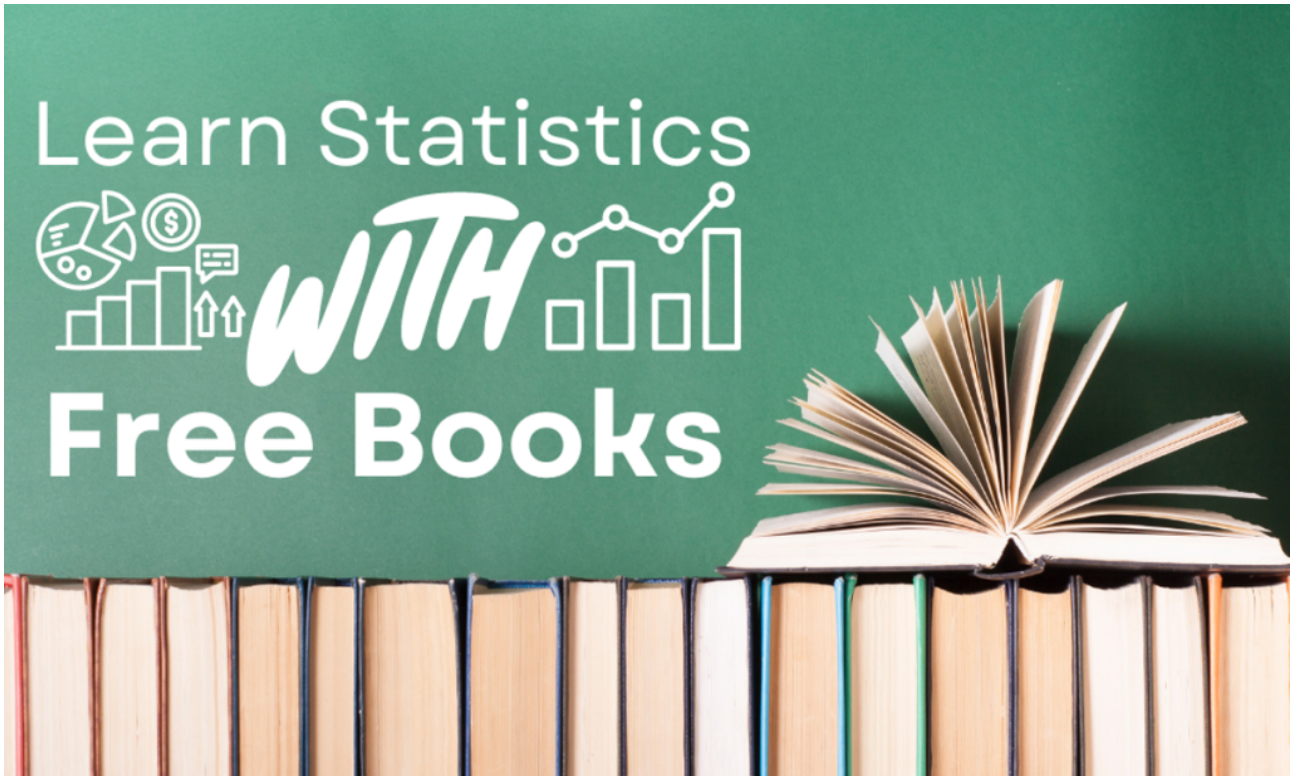
April 25, 2024

[5 Free Statistics Courses for Beginners]

April 19, 2024

5 MIT Statistics Courses That Are Free

April 18, 2024



5 Free Books to Learn Statistics

April 18, 2024

[How to Use the info() Method in Pandas]

April 12, 2024

How to Use pct_change() in Pandas

April 12, 2024