Understanding and Resolving the “NA/NaN/Inf in ‘y'” Error in R’s lm.fit Function

Authored by

Mohammed looti

October 30, 2025

One of the most frequent challenges faced by users performing statistical analysis in R involves handling missing or non-finite data points. When attempting to fit a linear regression model using the standard functions, you may abruptly encounter a detailed yet frustrating error message:

Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
NA/NaN/Inf in 'y'

This error specifically signals that the underlying fitting algorithm, **lm.fit**, has detected non-standard numeric values--either **NaN** (Not a Number) or **Inf** (Infinity)--within the response variable (y) or, less commonly, the predictor variables (x). While R's primary modeling functions are designed to gracefully manage standard **NA** (Not Available) missing values, the presence of **NaN** or **Inf** halts the process entirely. Understanding the subtle differences between these three types of non-finite values is the first step toward resolving this common statistical hurdle.

This guide will explore the technical reasons behind this failure and provide a clear, practical example demonstrating how to preprocess your data to ensure a successful model fit. We will focus on the most effective strategy: converting mathematically ambiguous values into a format R can interpret and exclude during model estimation.

Understanding the R Error: NA, NaN, and Inf

The core of this problem lies in R's nuanced distinction between various forms of non-finite data. In computational statistics, **NA**, **NaN**, and **Inf** each represent unique states, and R handles them with differing levels of tolerance, particularly within functions like **lm.fit**.

The **NA** value simply indicates a piece of data is missing or unavailable. When fitting a linear regression model, R's standard procedure (specifically, the default setting of `na.action='na.omit'`) is to remove any observation that contains an **NA** value in any of the variables used in the model. This listwise deletion allows the remaining, complete observations to be used to successfully estimate the model parameters. Crucially, the presence of **NA** values alone will not typically cause the `lm.fit` error.

In contrast, **NaN**, or Not a Number, represents the result of an undefined mathematical operation, such as dividing zero by zero. **Inf**, or Infinity, results from operations that exceed the limits of floating-point representation, such as dividing a non-zero number by zero. These values, defined under the IEEE 754 standard, are fundamentally different from **NA** because they represent mathematical singularities rather than simply missing data. The **lm.fit** routine, which relies on matrix algebra to solve the least squares problem, cannot incorporate these non-finite numbers into its calculations, leading directly to the reported error and termination of the fitting process.

The Core Mechanism of the lm.fit Function

To fully appreciate why these values cause a breakdown, it is helpful to understand the role of the **lm.fit** function. While most R users interact with the higher-level function `lm()`, it is **lm.fit** that performs the heavy computational lifting. The `lm()` function is essentially a wrapper: it prepares the data (creating the design matrix and handling factor variables), sets up the model formula, and then passes the resulting matrices (X for predictors, Y for response) to the optimized fitting engine, **lm.fit**.

The **lm.fit** function implements algorithms, often based on QR decomposition or similar techniques, to solve the matrix equation Ordinary Least Squares (OLS). For this matrix algebra to function correctly and provide stable, unique solutions, all elements within the input vectors and matrices must be finite, real numbers. The inclusion of **NaN** or **Inf** immediately violates this fundamental requirement.

If **NaN** or **Inf** were allowed to persist, they could propagate through the matrix operations, rendering the resulting coefficient estimates meaningless or causing numerical instabilities that crash the calculation entirely. Therefore, the error message serves as a strict warning, forcing the user to perform necessary data cleaning before the model fitting can proceed reliably. This strictness ensures the mathematical integrity of the model output, preventing potentially erroneous statistical conclusions based on non-finite input data.

Practical Demonstration: Reproducing the Error

To illustrate this issue clearly, let us construct a sample data set typical of real-world scenarios, where data collection or transformation processes have introduced problematic values. Suppose we are analyzing basketball statistics, attempting to predict points scored based on minutes played. Our data frame, however, contains a mix of standard missing values, non-numbers, and infinite results.

We begin by creating the data frame in R, intentionally injecting the critical values:

#create data frame with some NA, NaN, Inf values
df <- data.frame(minutes=c(4, NA, 28, 12, 30, 21, 14),
points=c(12, NaN, 30, Inf, 43, 25, 17))

#view data frame
df

minutes points
1 4 12

2 NA NaN
3 28 30
4 12 Inf
5 30 43
6 21 25
7 14 17

Upon reviewing this structure, we can identify several problematic observations. Row 2 contains `NA` in `minutes` and `NaN` in `points`. Row 4 contains a finite value in `minutes` but an `Inf` value in `points`. If we were only dealing with the `NA` in row 2, the `lm()` function would simply discard that row and continue.

However, when we proceed to attempt the linear regression model fit, using "minutes" as the predictor and "points" as the response, the presence of **NaN** and **Inf** immediately triggers the failure state, confirming the sensitivity of the **lm.fit** routine to these specific data types.

#attempt to fit regression model
lm(points ~ minutes, data=df)

Error in lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
NA/NaN/Inf in 'y'

The error message highlights that the failure is not due to the standard missing values (`NA`), but specifically due to the non-finite values (`NaN`/`Inf`) that cannot be processed by the underlying numerical solver. This confirms that the data preparation step must explicitly target these non-standard values before any model fitting can be reliably executed.

Implementing the Solution: Converting Invalid Values

The most robust and straightforward fix for this error involves a targeted data cleaning step: converting all instances of **NaN** and **Inf** into the standard **NA** missing value marker. By doing this, we effectively normalize all non-finite data points into a format that the `lm()` function knows how to handle gracefully (by omission).

The conversion process requires identifying cells that are either explicitly NaN or that represent Infinity. In R, we can use a combination of logical operators to achieve this identification across the entire data frame simultaneously. We must check for standard missingness (`is.na(df)`) while also checking for explicit Infinity representations. Note that `is.na()` conveniently returns TRUE for both `NA` and `NaN` values, but we must explicitly check for `Inf` values using a relational operator.

The following R command effectively identifies all cells containing **NA**, **NaN**, or **Inf**, and overwrites them with **NA**. This ensures that the original `NA` values are preserved as `NA`, while the problematic **NaN** and **Inf** values are converted to the same acceptable status.

#Replace NaN & Inf with NA
df = NA

#view updated data frame
df

minutes points
1 4 12
2 NA NA
3 28 30
4 12 NA
5 30 43
6 21 25
7 14 17

The updated data frame now shows **NA** where **NaN** and **Inf** previously resided. Specifically, the value in `points` for Row 2 (originally `NaN`) and Row 4 (originally `Inf`) are now standard `NA`. The original `NA` in `minutes` for Row 2 remains `NA`. This transformation renders the data frame suitable for the statistical fitting routine.

Verifying the Successful Linear Model Fit

With the data frame sanitized, we can now confidently re-attempt the linear regression analysis. The `lm()` function will now process the data, identify the observations containing **NA** in either the predictor or response variable (Rows 2 and 4), omit those observations from the fitting process, and successfully execute the **lm.fit** algorithm on the remaining complete observations.

The successful output confirms that the data preparation step effectively addressed the numerical stability issues inherent in the presence of **NaN** and **Inf**.

#fit regression model
lm(points ~ minutes, data=df)

Call:
lm(formula = points ~ minutes, data = df)

Coefficients:

(Intercept) minutes
5.062 1.048

The output displays the estimated coefficients of the regression model, indicating a successful fit. The intercept is estimated at 5.062, and the coefficient for `minutes` is 1.048. We receive no error because the **lm.fit** function was passed a clean matrix containing only finite, real numbers, allowing the mathematical solution for the least squares minimization to be achieved without interruption. This critical step ensures that the statistical inference derived from the model is based on mathematically sound inputs.

Best Practices for Data Cleaning in Statistical Modeling

While converting **NaN** and **Inf** to **NA** is a quick and effective fix for the `lm.fit` error, a robust data workflow requires understanding *why* these values appear and implementing systemic data cleaning checks. Non-finite values typically arise from three primary sources:

Data Entry Errors: Manual input mistakes can sometimes result in text being interpreted as `NaN` or `Inf` if the data type conversion is faulty, though this is less common in modern data pipelines.

Mathematical Operations: The most common source is division by zero (resulting in `Inf`) or operations where the result is undefined (e.g., `0/0` or `log(0)`, resulting in `NaN`). This often occurs during feature engineering when creating new variables based on ratios.

Data Transfer Limits: Rarely, data imported from external systems might contain values that exceeded the storage limits of the original source, which R then interprets as Infinity.

Before fitting any complex statistical model, it is a recommended practice to include explicit checks for these non-finite values, regardless of whether you anticipate using the **lm.fit** function. A comprehensive data hygiene protocol should involve using functions like `is.nan()` and `is.infinite()` early in the data preparation script to identify and handle these outliers, ensuring data integrity across all subsequent analyses. Proper handling minimizes unexpected crashes and guarantees that the statistical results reflect the data accurately.

Further Exploration and Resources

Mastering data preparation is essential for advanced statistical modeling in R. Understanding the nuances of missing data handling, especially the distinction between NA, NaN, and Inf, allows analysts to build more stable and reliable models. For those interested in deeper statistical methods, exploring imputation techniques--where missing values are estimated rather than simply omitted--can be the next step once non-finite values are successfully converted to standard **NA**.

To continue strengthening your skills in data cleaning and error resolution within R, consider reviewing documentation on advanced data manipulation packages, such as `dplyr` and `tidyr`, which offer powerful tools for efficiently identifying and transforming problematic data elements across large data sets. Always prioritize data quality before execution of any computationally intensive statistical procedure.