Learning the `relevel()` Function in R: A Guide for Regression Analysis with Categorical Variables

Authored by

Mohammed looti

November 13, 2025

The Role of Categorical Variables in Linear Regression

Linear regression stands as a cornerstone of statistical modeling, widely employed in research and data science to establish and quantify the mathematical relationship between a response variable and one or more predictor variables. This technique allows analysts to rigorously model how changes in inputs influence outcomes, offering essential insights into association and potential causality. When constructing these models, we encounter various data types, ranging from continuous measurements, such as height or temperature, to discrete groupings that classify observations.

A unique challenge arises when integrating a categorical variable--data that sorts observations into distinct, non-numerical groups--into a regression framework. Common examples include experimental treatment groups, geographical regions, or different educational attainment levels. Since these variables lack inherent quantitative magnitude, they cannot be directly included in the regression equation. Instead, they must be converted into a numerical format, typically through a process known as dummy coding or the creation of indicator variables, making them suitable for analysis.

Once a categorical predictor is introduced and coded, the resulting model coefficients do not represent the direct value of a given category. Rather, they quantify the expected average difference in the response variable relative to a specific reference point established within the model. This crucial reference point is universally referred to as the **baseline level**. Understanding the mechanism by which this baseline is chosen and mastering the ability to manipulate it are fundamental skills necessary for accurate interpretation of complex statistical findings, particularly when comparing the effects of various groups.

Defining and Interpreting the Regression Baseline

Within statistical computing environments, such as R, when a categorical variable--stored internally as a factor--is entered into a regression equation, the software automatically designates one of its levels to serve as the baseline or reference group. The effects of all other levels within that variable are then mathematically calculated and compared against this established baseline. Consequently, the regression coefficients associated with the non-baseline categories represent the anticipated change in the outcome variable when transitioning from the baseline group to that specific category, assuming all other predictors remain constant.

R's default selection mechanism typically chooses the level that appears first, either alphabetically or numerically, to function as this baseline. While this automatic process simplifies initial model fitting, it frequently results in model summaries that are not optimally aligned with the researcher's specific interpretational goals or hypotheses. For instance, if a study involves comparing multiple experimental treatments against a well-defined standard control group, the control group should

logically serve as the baseline, irrespective of its alphabetical position among the factor levels.

The selection of the baseline significantly alters the numerical value and the substantive meaning of the individual coefficients displayed in the summary table. It is vital to note, however, that while interpretation changes, the overall predictive power, the statistical fit, and the residuals of the entire linear model remain completely unchanged. If the automated baseline choice hinders clarity--perhaps you need a particular group to anchor the comparison--manual intervention is required. This is precisely the scenario where the intuitive and powerful **relevel()** function in R programming becomes essential for tailored statistical analysis.

Introducing the R Function: relevel() Syntax and Purpose

The **relevel()** function is specifically engineered to reorganize the internal order of levels within a factor variable, ensuring that a desired level is moved into the premier position. Because R's regression functions rely on the first level of a factor as the default baseline, employing **relevel()** effectively mandates the chosen reference group for any subsequent statistical modeling. This critical manipulation grants the researcher explicit control over the comparison point, leading to a much clearer and more meaningful interpretation of the model's structure and the estimated effects.

The core usage of **relevel()** is remarkably simple, requiring only two primary inputs. A fundamental prerequisite for its application is that the variable must be a factor; if your categorical data is currently stored as numeric or character data types, you must first convert it using the `as.factor()` function before attempting to apply **relevel()**. Failing to adhere to this data type requirement will result in an error, as the function is designed solely for manipulating factor levels.

You can utilize the **relevel()** function in R using the following concise syntax:

relevel(x, ref)

The function's arguments are defined as follows:

x: The factor variable whose internal level order you intend to modify. This must be a properly structured, unordered factor object.
ref: The reference level, specified as the precise level you wish to designate as the new baseline. This input must be provided as a character string that exactly matches the level name (e.g., 'A', '3', or 'Control Group').

The subsequent practical example meticulously details the application of the **relevel()** function in a real-world linear regression context, clearly illustrating how changing the baseline dramatically alters the interpretation of the resulting regression coefficients without affecting the underlying statistical fit.

Setting Up the Practical Demonstration in R

To effectively illustrate the utility and necessity of **relevel()**, we will construct a hypothetical dataset derived from 12 basketball players. This dataset captures key performance indicators alongside details of their respective training regimens. Our primary objective is to determine how the total hours dedicated to practice and the specific type of training program utilized influence the total points scored by each player.

Our data frame comprises three essential variables: **points scored** (our quantitative response variable), **hours spent practicing** (a continuous predictor), and the **training program used** (a categorical variable distinguished by three separate levels: 1, 2, and 3). We hypothesize that both the time investment (hours) and the programmatic structure (program type) are significant determinants of player performance.

The initial step requires constructing and initializing the data frame in R, ensuring the data is correctly structured before proceeding with any analytical modeling. The following R code snippet creates the data frame, labeled `df`, and displays its contents to verify proper initialization and data entry:

```
#create data frame
df <- data.frame(points=c(7, 7, 9, 10, 13, 14, 12, 10, 16, 19, 22, 18),
hours=c(1, 2, 2, 3, 2, 6, 4, 3, 4, 5, 8, 6),
program=c(1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3))

#view data frame
df

points hours program
1 7 1 1
2 7 2 1
3 9 2 1
4 10 3 1
5 13 2 2
6 14 6 2
7 12 4 2
8 10 3 2
9 16 4 3
10 19 5 3
11 22 8 3
12 18 6 3
```

We aim to fit the classic linear regression model defined below, quantifying the influence of practice time and program type on performance:

points = β0 + β1hours + β2program

In this formulation, the `program` variable is clearly a categorical variable capable of assuming one of three distinct integer values: 1, 2, or 3. Critically, before executing the model fitting process, we must explicitly ensure that the `program` column is recognized as a factor. This conversion instructs R to appropriately handle the variable using indicator coding, allowing for correct interpretation of the group effects.

Analyzing the Default Model Output

Our first analytical step involves fitting the linear regression model using R's standard `lm()` function without specifying a reference level for the `program` variable. This crucial step allows us to document R's default behavior, which, based on the numerical ordering of the factor levels, will automatically designate Program 1 as the reference group. We begin by converting the `program` column to the required factor type.

The following comprehensive code block performs the necessary data conversion, fits the linear model, and subsequently displays the detailed summary statistics:

```
#convert 'program' to factor
df$program <- as.factor(df$program)

#fit linear regression model
fit <- lm(points ~ hours + program, data = df)

#view model summary
summary(fit)
```

```
Call:
lm(formula = points ~ hours + program, data = df)

Residuals:
Min 1Q Median 3Q Max
-1.5192 -1.0064 -0.3590 0.8269 2.4551

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 6.3013 0.9462 6.660 0.000159 ***
hours 0.9744 0.3176 3.068 0.015401 *
```

```
program2 2.2949 1.1369 2.019 0.078234 .
program3 6.8462 1.5499 4.417 0.002235 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.403 on 8 degrees of freedom
Multiple R-squared: 0.9392, Adjusted R-squared: 0.9164
F-statistic: 41.21 on 3 and 8 DF, p-value: 3.276e-05
```

Upon careful examination of the **Estimate** column in the regression output table, we can clearly identify distinct coefficients provided for `program2` and `program3`. The conspicuous absence of a coefficient specifically labeled `program1` confirms that R has automatically designated **Program 1** as the **baseline level** for this categorical variable.

Interpreting this default output reveals that the coefficient for `program2` (2.2949) indicates that, holding practice hours constant, players participating in program 2 score 2.2949 more points, on average, compared to those in program 1. Similarly, the coefficient for `program3` (6.8462) suggests that players in program 3 score 6.8462 more points than those in program 1. While mathematically sound, this interpretation forces all comparisons to be relative to Program 1. If the research focus requires comparing performance against, say, Program 3 (perhaps the highest-performing group), this default summary may be structurally inconvenient and less intuitive.

Manually Setting the Baseline with relevel()

Let us suppose that, for enhanced interpretational clarity, our goal is to specifically designate **Program 3** as the reference level. We might choose this if Program 3 represents a gold standard or a new experimental benchmark, and we wish to measure the performance deficits or differences of Programs 1 and 2 directly against this superior group.

To achieve this shift in reference, we must strategically employ the **relevel()** function immediately after ensuring the variable is a factor, but prior to executing the `lm()` function. We supply the factor variable (`df$program`) along with the desired new reference level, specifying it using the `ref` argument, which in this instance is the character string '3'. This operation internally reorders the factor's levels, ensuring that '3' is moved to the first position.

The following refined syntax demonstrates the necessary steps: converting the variable to a factor, specifying Program 3 as the new baseline using **relevel()**, and subsequently refitting the linear regression model to incorporate this change:

```
#convert 'program' to factor
df$program <- as.factor(df$program)
```

```
#specify that program3 should be used as baseline level
df$program <- relevel(df$program, ref='3')

#fit linear regression model
fit <- lm(points ~ hours + program, data = df)

#view model summary
summary(fit)

Call:
lm(formula = points ~ hours + program, data = df)

Residuals:
Min 1Q Median 3Q Max
-1.5192 -1.0064 -0.3590 0.8269 2.4551

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 13.1474 1.9563 6.721 0.00015 ***
hours 0.9744 0.3176 3.068 0.01540 *
program1 -6.8462 1.5499 -4.417 0.00223 **
program2 -4.5513 1.1777 -3.864 0.00478 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.403 on 8 degrees of freedom
Multiple R-squared: 0.9392, Adjusted R-squared: 0.9164
F-statistic: 41.21 on 3 and 8 DF, p-value: 3.276e-05
```

Interpreting the Revised Model Coefficients

Upon reviewing the output of the newly fitted model, the successful execution of **relevel()** is immediately evident. The **Estimate** column now explicitly displays coefficients for `program1` and `program2`, while the term `program3` is omitted. This structural change confirms that **Program 3** has been correctly established as the **baseline level**, meaning all subsequent interpretations must be understood relative to this specific group.

It is important to notice that the global model statistics, including the R-squared value, the Residual standard error, and the F-statistic, remain mathematically identical to the previous model output. This consistency underscores a key principle of regression analysis: changing the baseline only affects how the categorical variable's total effect is partitioned and presented across its individual

levels, not the overall goodness of fit of the model to the observed data.

However, the interpretation of the individual group coefficients has fundamentally changed. The coefficient for `program1` is now -6.8462. This indicates that, controlling for practice hours, players in program 1 score 6.8462 points fewer than those in the new baseline group, program 3. Similarly, the coefficient for `program2` is -4.5513, suggesting that program 2 participants score 4.5513 points less than the program 3 participants. By utilizing **relevel()**, we have successfully generated a model summary that directly addresses the research question by comparing all groups against our desired high-performance benchmark.

Additional Resources for Data Analysis in R

The ability to manipulate factor levels precisely is a necessary skill for conducting rigorous statistical analysis in R. The following tutorials explore other common tasks and help deepen your mastery of statistical modeling techniques and data preparation:

How to Handle Missing Data in R
Conducting ANOVA with Post-Hoc Tests in R
Understanding the Output of Generalized Linear Models

<!--

Featured Posts

-->