How to Run a Linear Regression on a Data Frame Subset in R

Authored by

stats writer

November 20, 2025

Performing statistical analysis on large datasets often requires focusing only on specific observations that meet certain criteria. In the R programming environment, the process of fitting a linear regression model frequently involves using only a data frame subset.

While some users might attempt to slice the data frame using bracket notation (e.g., `df`) before passing it to the `lm()` function, the most efficient and recommended approach is utilizing the built-in **subset** argument directly within the function call itself. This method streamlines the code and improves readability, ensuring that the model fitting process is executed only on the required rows without creating intermediate data structures.

For instance, an initial, less efficient attempt might look like `lm(y ~ x, data=df)`. However, the `lm()` function is specifically designed to handle this filtering internally, allowing for powerful conditional selection based on variable values within the data frame.

Understanding the subset Argument in lm()

The **subset** argument provides a clean and powerful mechanism for defining the criteria that observations must meet to be included in the regression model fitting process. It accepts a logical expression that evaluates to `TRUE` or `FALSE` for each row of the input data frame. Only rows yielding `TRUE` are ultimately used for calculating the coefficients.

You can use the **subset** argument to only use a subset of a data frame when using the `lm()` function to fit a regression model in R. This approach ensures that your analytical focus remains sharp and targeted on the relevant population within your dataset. The expression must reference columns that exist within the specified `data` argument.

fit <- lm(points ~ fouls + minutes, data=df, subset=(minutes>10))

This particular example demonstrates a model fitting using **points** as the primary response variable and **fouls** and **minutes** as the corresponding predictor variables. The syntax is concise and immediately tells the reader that the analysis is restricted.

Crucially, the **subset** argument specifies that only the rows in the data frame where the **minutes** variable is strictly greater than 10 should be included when performing the model estimation. This is invaluable when analyzing subgroups, removing outliers, or enforcing constraints based on experimental design. The following example provides a comprehensive, hands-on demonstration of how to implement and verify this powerful syntax in practice.

Example: Setting up Data for Conditional Linear Regression

To illustrate the functionality of the **subset** argument, let us first establish a realistic data frame in

R. This synthetic dataset contains critical performance metrics--minutes played, total fouls committed, and total points scored--for ten hypothetical basketball players. This simple structure allows us to easily visualize the effects of our subsetting criteria.

```
#create data frame
df <- data.frame(minutes=c(5, 10, 13, 14, 20, 22, 26, 34, 38, 40),
fouls=c(5, 5, 3, 4, 2, 1, 3, 2, 1, 1),
points=c(6, 8, 8, 7, 14, 10, 22, 24, 28, 30))
```

```
#view data frame
df
```

```
minutes fouls points
1 5 5 6
2 10 5 8
3 13 3 8
4 14 4 7
5 20 2 14
6 22 1 10
7 26 3 22
8 34 2 24
9 38 1 28
10 40 1 30
```

Our objective is to fit a specific multiple linear regression model to understand the relationship between points scored (response) and minutes played and fouls committed (predictors). The theoretical model we intend to estimate is defined by the following equation:

points = β0 + β1(minutes) + β2(fouls)

For analytical purposes, suppose we hypothesize that the relationship between these variables only holds true, or is only relevant, for players who have played a significant amount of time--specifically, those whose **minutes** variable is greater than 10. This requirement necessitates using the **subset** argument to filter the data prior to coefficient estimation.

Applying Simple Conditional Filtering using subset

To implement this conditional analysis, we invoke the `lm()` function, ensuring that the **subset** argument contains the logical test `(minutes > 10)`. This ensures that only the observations where the condition evaluates to true are passed into the estimation algorithm, isolating the impact of longer playing times on performance metrics.

#fit multiple linear regression model (only for rows where minutes>10)
fit <- lm(points ~ fouls + minutes, data=df, subset=(minutes>10))

#view model summary
summary(fit)

Call:
lm(formula = points ~ fouls + minutes, data = df, subset = (minutes >
10))

Residuals:
3 4 5 6 7 8 9 10
1.2824 -2.5882 2.2000 -1.9118 2.3588 -1.7176 0.1824 0.1941

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -11.8353 4.9696 -2.382 0.063046 .
fouls 1.8765 1.0791 1.739 0.142536
minutes 0.9941 0.1159 8.575 0.000356 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.255 on 5 degrees of freedom
Multiple R-squared: 0.9574, Adjusted R-squared: 0.9404
F-statistic: 56.19 on 2 and 5 DF, p-value: 0.0003744

The output summary confirms that the model was fitted using the specific subsetting criteria defined in the `Call:` section. Interpreting these results, we observe that the **minutes** variable remains highly significant (p-value of 0.000356) for this subgroup of players, with an estimated coefficient of 0.9941. This suggests that for players playing more than 10 minutes, each additional minute played is associated with approximately one extra point scored, holding fouls constant.

Verifying the Sample Size with nobs()

When applying subsetting, it is critical to confirm that the correct number of observations were utilized in the analysis. This step ensures that the logical conditions applied in the **subset** argument functioned as intended. We can easily verify the effective sample size of the fitted regression model object using the **nobs()** function in R.

#view number of observations used to fit model
nobs(fit)

8

The output confirms that **8** rows from the original data frame were used to fit this specific regression model. Comparing this number to our original dataset, we can manually check the rows where `minutes > 10`. Rows 1 and 2 (with 5 and 10 minutes, respectively) are excluded, leaving exactly 8 rows (rows 3 through 10).

This verification step is especially important in more complex scenarios involving missing data or multiple conditions, as it provides a necessary check on data integrity. The ability to use **nobs()** immediately after fitting the model makes the workflow robust and transparent, ensuring that the model accurately reflects the intended subset of the population.

Applying Multiple Logical Conditions for Refined Filtering

The power of the **subset** argument is further amplified when utilizing multiple logical conditions combined with Boolean operators. Common operators include the logical AND (`&`) and the logical OR (`|`). Using these allows researchers to define highly specific, complex filters for their analysis.

For instance, we might want to narrow our focus even further: fitting a regression model using only the rows in the data frame where **minutes** is greater than 10 *and* **fouls** is less than 4. This criteria targets high-minute, low-foul players.

#fit multiple linear regression model (only where minutes>10 & fouls<4)
fit <- lm(points ~ fouls + minutes, data=df, subset=(minutes>10 & fouls<4))

#view number of observations used to fit model
nobs(fit)

7

From this output, we observe that **7** rows from the data frame were utilized to fit this particular model. One of the previously included rows (Row 4: 14 minutes, 4 fouls) was excluded because it failed the secondary condition (fouls < 4). The ability to define these intricate constraints directly within the `lm()` call eliminates the need for separate data manipulation steps, making the entire analytical script more direct and traceable.

Advanced Subsetting: Handling Missing Data (NA values)

One of the often-overlooked advantages of using the **subset** argument is its interaction with missing values (`NA`). In R, when a logical condition evaluates to `NA`, the corresponding row is

typically excluded from the analysis by default when using the `lm()` function. This behavior is usually desirable because we cannot definitively determine if an observation with missing data meets a logical filtering requirement.

When filtering a data frame manually before feeding it to `lm()`, handling NAs requires explicit use of functions like `is.na()`, adding complexity. By contrast, the **subset** argument automatically integrates with R's default behavior regarding logical vectors containing NAs, contributing to cleaner, safer model generation.

Furthermore, the `lm()` function also includes the **na.action** argument, which determines how missing values are handled in the variables used in the model formula. When combined with **subset**, these two arguments provide comprehensive control over which observations contribute to the final coefficients of the regression model, ensuring reliable and interpretable results.