Fix in R: plot.new has not been called yet

Authored by

Mohammed looti

November 2, 2025

Working with the graphics capabilities of R is typically straightforward, but users occasionally encounter a specific, confusing error message that halts plotting operations. This error indicates that the graphical environment has not been properly initialized.

The error message you might see in your R console is as follows:

Error in plot.xy(xy.coords(x, y), type = type, ...) :
plot.new has not been called yet

This message arises when you attempt to use a low-level plotting function--one designed only to add elements to an existing plot--without first creating the necessary graphical canvas. Essentially, R needs an initialized plot space before it can draw supplementary features like additional lines or points.

Understanding the distinction between high-level and low-level plotting functions in R is key to resolving this issue. The following sections provide a detailed explanation of the error's root cause and present practical, code-based solutions.

Understanding the "plot.new has not been called yet" Error

The core of this problem lies in the design of R's base graphics system. Plotting functions in R are categorized based on their role: **high-level** functions initiate a new graph, while **low-level** functions modify or enhance an existing one.

When any high-level plotting command, such as `plot()`, `hist()`, or `boxplot()`, is executed, it performs an internal action: it calls the function plot.new. This internal function is responsible for opening the graphics device (the window where the plot appears), setting up the coordinate system, and establishing the plotting parameters. It effectively creates the blank canvas required for drawing.

Low-level functions, such as `lines()`, `abline()`, `points()`, or `text()`, are designed purely for superimposition. They assume that the coordinate system and plotting region have already been defined by a preceding call to a high-level function. If a low-level function is executed first, it finds no active plot device or coordinate system, resulting in the "plot.new has not been called yet" error.

To avoid this error, the workflow must always begin with a high-level function call that initializes the graphical environment. This establishes the context needed for subsequent modifications.

Example 1: Resolving the Error When Using the lines() Function

The lines() function is a common low-level function used to add connected line segments to a

graph. It is frequently employed when adding a fitted model, such as a regression line, to a scatterplot.

Consider the following scenario where we attempt to fit a polynomial regression model and immediately plot the fitted line using `lines()`:

```
#create data
df <- data.frame(x=c(1, 2, 2, 3, 5, 6, 8, 8, 9, 9, 10, 11, 12, 15, 15),
y=c(2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10, 16, 19, 28))

#fit polynomial regression model
model <- lm(y~poly(x, 2), data=df)

#define new sequence of x-values
new_x <- seq(min(df$x), max(df$y))

#attempt to plot fitted regression line (ERROR OCCURS HERE)
lines(new_x, predict(model, newdata = data.frame(x=new_x)))
```

```
Error in plot.xy(xy.coords(x, y), type = type, ...) :
plot.new has not been called yet
```

As demonstrated above, the code fails because the lines() function is executed before any high-level plotting function has called plot.new. We must first establish the plot's boundaries and display the raw data points before attempting to draw the fitted line over them.

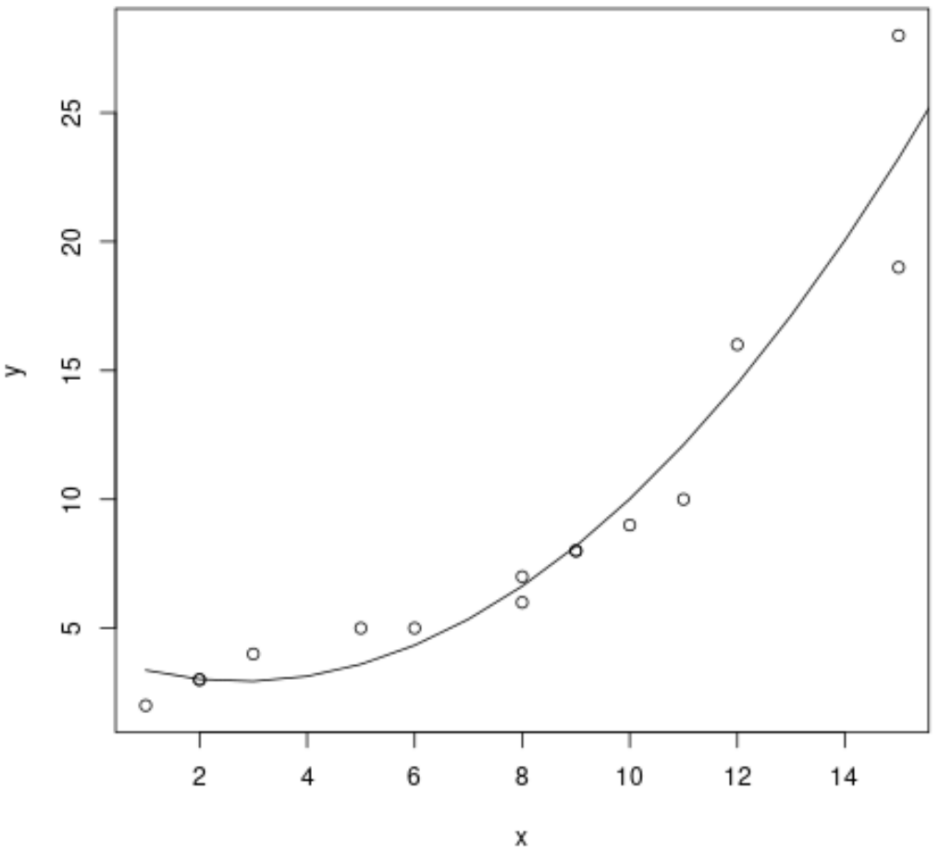
The solution involves inserting a high-level plotting command--in this case, `plot()`--immediately before the call to lines(). This ensures the graphical device is properly initialized and the coordinate system is set up based on the range of the data.

The corrected code sequence is as follows:

```
#create data
df <- data.frame(x=c(1, 2, 2, 3, 5, 6, 8, 8, 9, 9, 10, 11, 12, 15, 15),
y=c(2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10, 16, 19, 28))

#fit polynomial regression model
model <- lm(y~poly(x, 2), data=df)

#define new sequence of x-values
new_x <- seq(min(df$x), max(df$y))

#create scatterplot of x vs. y values (INITIALIZES THE PLOT)
```

```
plot(y~x, data=df)

#attempt to plot fitted regression line (SUCCESS)
lines(new_x, predict(model, newdata = data.frame(x=new_x)))
```

By executing `plot(y~x, data=df)` first, we successfully generate the base scatterplot. The subsequent call to lines() then finds an existing plot canvas and correctly draws the fitted curve, producing the desired output:



Example 2: Resolving the Error When Using the abline() Function

Another frequently used low-level function that triggers this error is abline(). This function is specifically designed to add straight lines--horizontal, vertical, or sloped--to an existing plot. Since its entire purpose is additive, it does not contain the necessary internal call to initialize the plotting device.

Suppose we want to create a scatterplot of our data and immediately superimpose a horizontal reference line at y=10:

#create data

```
df <- data.frame(x=c(1, 2, 2, 3, 5, 6, 8, 8, 9, 9, 10, 11, 12, 15, 15),
y=c(2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10, 16, 19, 28))

#attempt to add horizontal line at y=10 (ERROR OCCURS HERE)
abline(a=10, b=0, lwd=2)
```

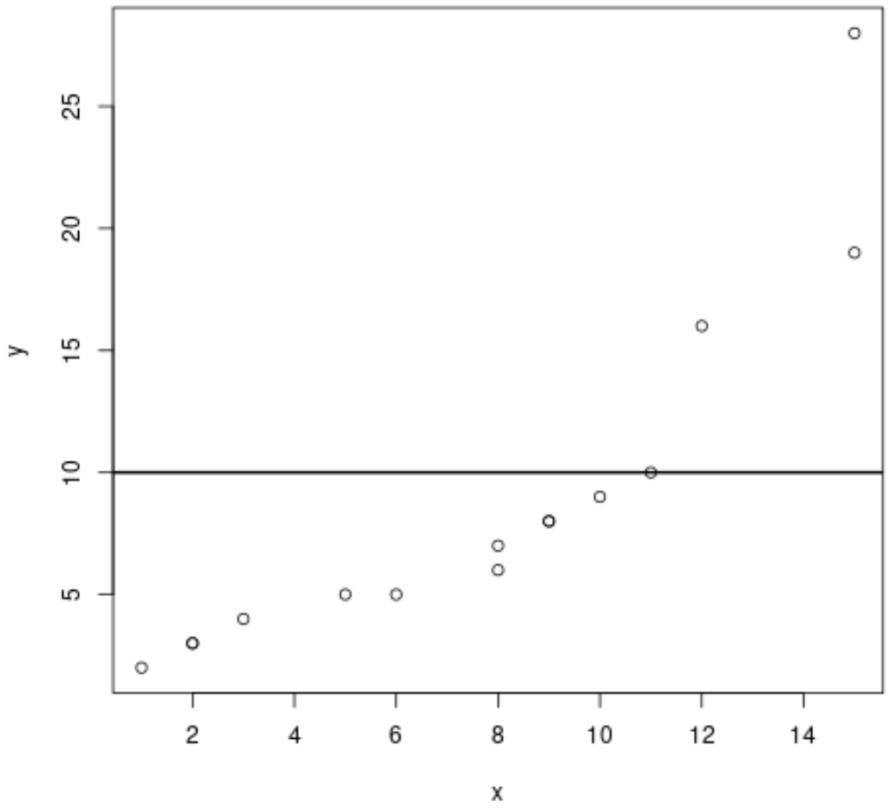
```
Error in plot.xy(xy.coords(x, y), type = type, ...) :
plot.new has not been called yet
```

The error occurs because abline(), when called first, has no graphical context to draw upon. R doesn't know the range of the x and y axes, nor does it have an open device window. It requires the base `plot()` command to define these essential parameters.

To successfully add the horizontal line, we must first use a high-level function, such as `plot()`, to create the foundational graph. Once the plot is generated, abline() can be used immediately afterward to draw the desired reference line.

```
#create data
df <- data.frame(x=c(1, 2, 2, 3, 5, 6, 8, 8, 9, 9, 10, 11, 12, 15, 15),
y=c(2, 3, 3, 4, 5, 5, 6, 7, 8, 8, 9, 10, 16, 19, 28))

#create scatterplot of x vs. y (INITIALIZES THE PLOT)
plot(y~x, data=df)

#add horizontal line at y=10 (SUCCESS)
abline(a=10, b=0, lwd=2)
```

By correctly sequencing the commands, we successfully generate the scatterplot followed by the horizontal line at y=10:



The Role of High-Level Plotting Functions in R

It is important for R users to recognize the specific functions that act as plot initializers. These are the commands that implicitly call plot.new and prepare the graphical device. While `plot()` is the most versatile and common initializer, several others perform the same crucial setup step.

High-level plotting functions typically include:

`plot()`: The default function for creating scatterplots, line plots, and general visualizations.
`hist()`: Used for creating histograms.
`boxplot()`: Used for generating box-and-whisker plots.
`image()`: Used for displaying matrices as images (heatmaps).
`curve()`: Used for plotting mathematical expressions.

If you are running a script or a function that requires low-level additions but you do not want to display any data points initially (for example, you only want to draw a frame), you can use the `plot()` function with specific arguments to create an empty canvas.

For instance, using `plot(x, y, type="n")` will initialize the plot device and set the correct axes based on the `x` and `y` data ranges, but it will suppress the plotting of the actual data points (`type="n"` stands for "no plotting"). This technique is often used when the user intends to draw all

elements using low-level functions like `points()` and lines() for maximum control.

Best Practices for Avoiding Graphics Initialization Errors

To maintain clean and functional R code, especially when dealing with base graphics, adherence to a few best practices can help prevent the "plot.new has not been called yet" error entirely.

Firstly, always mentally categorize your plotting commands. If a function's purpose is to modify or add to an existing visualization (e.g., adding labels, annotations, or fitting lines), ensure that a function designed to establish the plot's framework (a high-level function) precedes it. Never start a graphical sequence with a low-level function.

Secondly, when creating custom visualization functions in R, ensure that the very first graphical command executed within that function is a high-level initializer. If the function allows the user to optionally skip the data points, use the `type="n"` argument within the initial `plot()` call instead of omitting the `plot()` call entirely. This guarantees that the graphics context is always set up correctly.

Finally, remember that plotting systems outside of R's base graphics, such as **ggplot2**, manage the graphical state differently. While **ggplot2** is often preferred for complex visualizations due to its layered structure, the base R system requires explicit initialization, making careful command sequencing crucial for success.

Additional Resources

If you encounter other issues while programming in R, exploring documentation related to plotting device management and function hierarchies is highly recommended.

The following tutorials explain how to fix other common errors in R:

How to fix the "subscript out of bounds" error in R.
Understanding and fixing "object of type 'closure' is not subsettable" in R.
Troubleshooting the "replacement has length zero" error.