How to Set Axis Limits in ggplot2

Authored by

stats writer

December 23, 2025

Effective data visualization requires meticulous control over the display properties of a graph, particularly the axis ranges. In the world of R, the powerful package ggplot2 offers several methods to manipulate the minimum and maximum values of the X and Y axes. Understanding these methods is crucial because the choice of function dictates how the underlying data is treated, which can dramatically alter the visual representation and interpretation of the plot.

The primary mechanisms for establishing axis boundaries involve using functions derived from the underlying scaling system, such as `scale_x_continuous()` and `scale_y_continuous()`. These functions provide granular control, accepting arguments that define the specific minimum and maximum values desired for the respective axes. This ability to customize the range allows data analysts to focus the audience's attention on specific subsets of the data distribution, or to ensure uniformity across multiple related visualizations.

While the scaling functions offer deep customization, ggplot2 also provides convenient shortcut functions for quickly setting limits. However, it is paramount to recognize the fundamental difference between these two approaches: some functions filter out data points that fall outside the specified range before mapping, while others merely "zoom" the view without discarding observations. We will explore these distinctions in detail, providing a comprehensive guide to setting axis limits effectively and responsibly.

Core Methods for Defining Axis Ranges

When working with ggplot2, developers often turn to three primary functions for defining the visual boundaries of their plots. The first two, **`xlim()`** and **`ylim()`**, serve as wrappers for the underlying scaling functions and provide a straightforward way to declare the minimum and maximum extent of the X and Y axes, respectively. These functions are highly intuitive and are often the first choice for users seeking quick modifications to axis boundaries. For instance, if you require the X-axis to display only values between 10 and 50, applying **`xlim(10, 50)`** achieves this goal instantly.

However, it is critical to understand the mechanism by which **`xlim()`** and **`ylim()`** operate. When these functions are used, they modify the underlying scale of the data. Any data observation whose coordinate falls outside the newly defined range is effectively filtered out and removed from the dataset used to render the plot. This pre-processing step, while simplifying the visualization process, carries the inherent risk of introducing bias or hiding important contextual information, especially if the removal of data points is not clearly communicated to the audience.

In contrast to the data-filtering methods, the function **`coord_cartesian()`** offers a safer and often preferred alternative when the goal is purely visual modification, not data manipulation. This function operates on the coordinate system itself, zooming into a specified area of the plot space after all the data has been mapped and calculated. This means that observations falling outside

the visual limits remain part of the dataset, ensuring statistical calculations (like smooth lines or summaries) are based on the complete data, even if only a subset is visible. This distinction between filtering and zooming is fundamental to producing statistically sound visualizations in R.

`xlim()`: Specifies the lower and upper limit of the X-axis, resulting in the removal of observations outside this range.
`ylim()`: Specifies the lower and upper limit of the Y-axis, resulting in the removal of observations outside this range.
`coord_cartesian()`: Specifies the limits for the X-axis and Y-axis without dropping observations; it only changes the visual extent (zooming).

Understanding Data Removal in xlim() and ylim()

The core difference between the scaling functions (like `xlim()` and `ylim()`) and the coordinate function (`coord_cartesian()`) lies in the sequence of operations within the ggplot2 pipeline. When functions like `xlim()` are called, they interfere early in the plotting process, specifically during the scaling stage. This means that any data point (row) that fails the limit criteria is immediately dropped from the data frame passed to the subsequent geometric layers (like points, lines, or bars). This approach is functionally equivalent to applying a manual filter to the original dataset before passing it to the `ggplot()` call.

This data removal mechanism has significant practical implications. For simple scatterplots, the primary consequence is the visual disappearance of points outside the limits, often accompanied by a standard warning message from R indicating how many rows were removed. However, the impact is much greater when utilizing complex geoms or statistical summaries. For instance, if you apply a linear regression line using `geom_smooth(method = 'lm')`, setting limits via `xlim()` will cause the regression line to be calculated only using the remaining, truncated subset of data. This can lead to a regression line that poorly represents the relationship observed in the full dataset, potentially misleading the viewer.

Therefore, `xlim()` and `ylim()` should be reserved for cases where you genuinely wish to filter the data used in the analysis, or when the statistical calculation itself should only consider the values within the specified range. If the intent is merely to adjust the viewport--to zoom in or out on a dense area of the plot while maintaining the integrity of underlying statistical computations derived from the complete data--the use of `coord_cartesian()` is mandatory to avoid unintended data loss and inaccurate graphical representations.

The Non-Destructive Approach: coord_cartesian()

For scenarios where maintaining the integrity of the full dataset is paramount, `coord_cartesian()` is the essential function. This method operates at the coordinate level, which is the final stage of

the ggplot2 plotting pipeline, occurring after the data has been scaled and all geometric objects have been calculated. By specifying the `xlim` and `ylim` arguments within `coord_cartesian()`, the user effectively tells R to clip the rendered output to a specific rectangular region without modifying the mapped data points themselves.

The primary advantage of this "zooming" mechanism is that all statistical layers remain accurate. For example, if a plot includes a density estimate or a box plot, those summary statistics are computed using all available observations, even those that lie outside the visual limits defined by `coord_cartesian()`. This is critical for academic and professional reporting where statistical summaries must reflect the entire sample rather than a truncated subset created solely for visual purposes.

Furthermore, `coord_cartesian()` provides the flexibility to set both X and Y axis limits simultaneously within a single function call, improving code readability and efficiency. It accepts the arguments `xlim` and `ylim`, expecting vectors (usually generated by `c()`) defining the lower and upper bounds. When applied, this function ensures that the resulting plot is a faithful, albeit clipped, visualization of the fully processed data structure, eliminating the persistent warning messages associated with data removal that accompany `xlim()` and `ylim()`.

Demonstrating Axis Limit Control using the mtcars Dataset

To illustrate the practical application and key differences between these methods, we will use the standard built-in mtcars dataset available in R. This dataset contains information on 32 automobiles, including variables such as miles per gallon (`mpg`) and weight (`wt`). Our base plot will be a simple scatterplot mapping `mpg` to the X-axis and `wt` to the Y-axis. This initial visualization provides a baseline against which we can compare the effects of applying various limit functions.

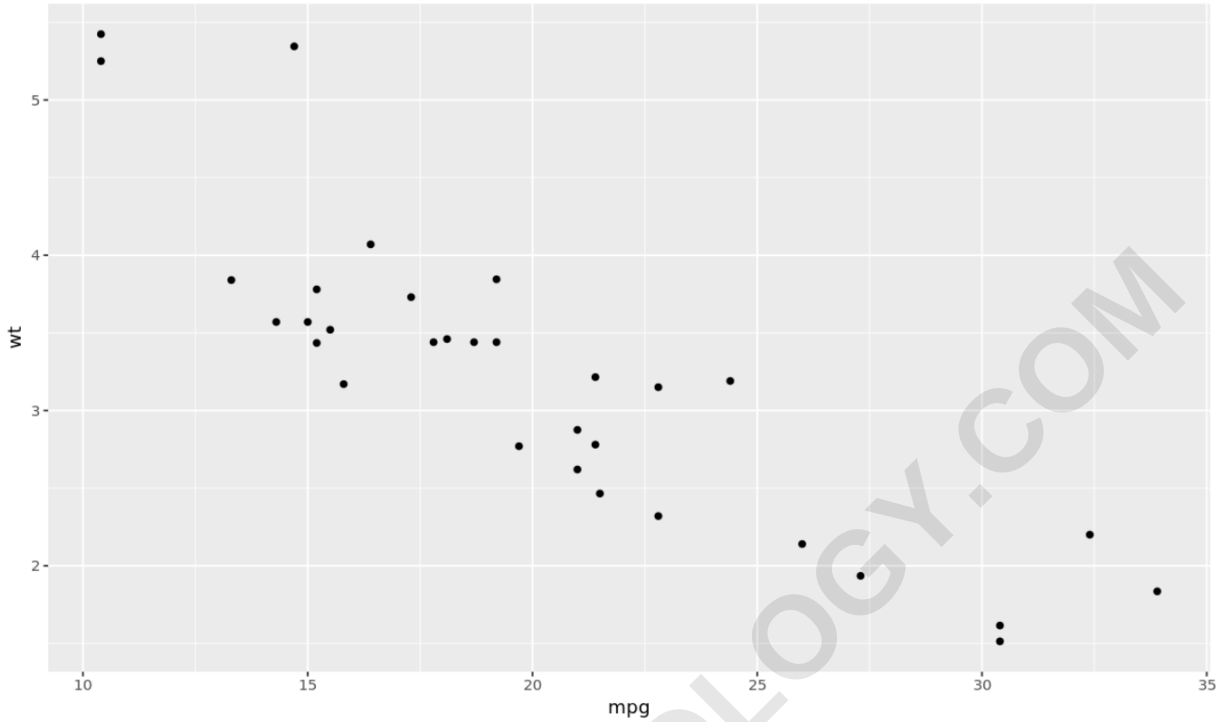
Before creating the base plot, we must ensure the ggplot2 library is loaded. The following code snippet loads the necessary package and generates the default scatterplot, showing the full range of the data points for comparison in subsequent examples.

```
# Load the ggplot2 package
library(ggplot2)

# Create the foundational scatterplot using mtcars data
ggplot(mtcars, aes(mpg, wt)) +
geom_point()
```

As depicted in the initial visualization below, the X-axis (`mpg`) currently ranges from approximately 10 to 35, and the Y-axis (`wt`) ranges from about 1.5 to 5.5. We will now proceed to manipulate these default ranges using the previously discussed functions, paying close attention to the visual

results and any accompanying warnings.



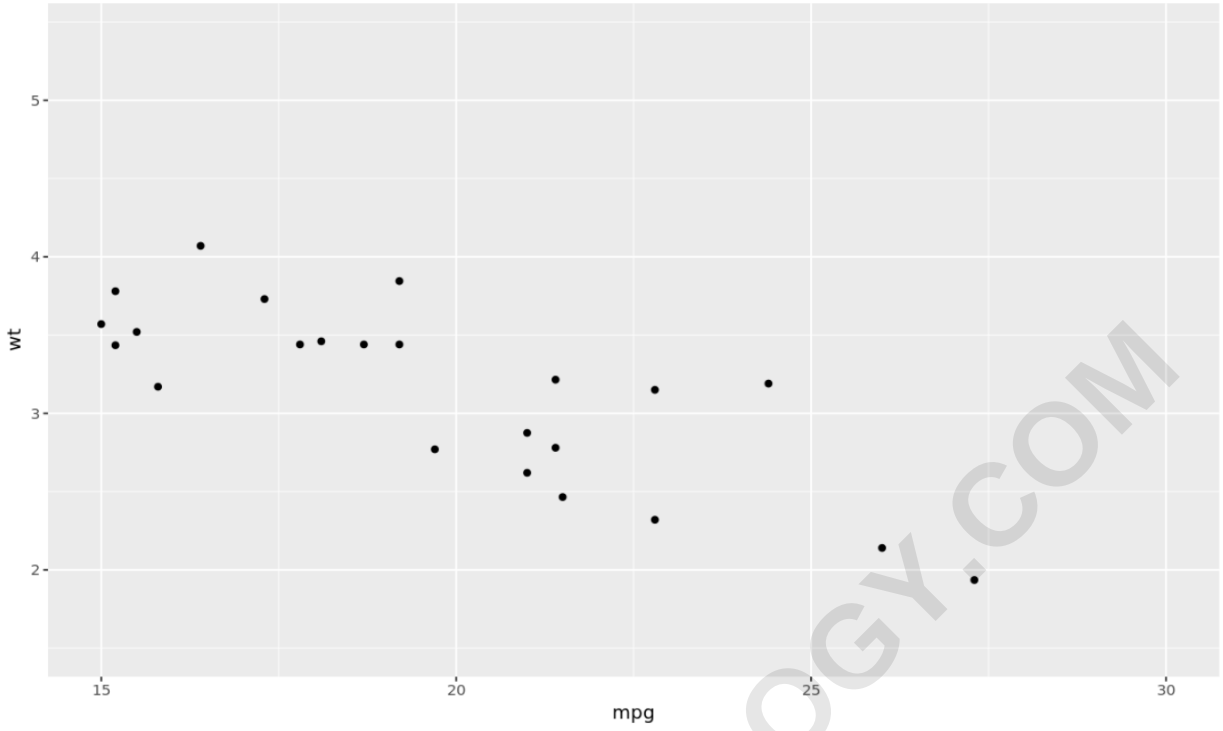
Example 1: Setting X-Axis Limits Using xlim()

The `xlim()` function is the most direct way to specify a minimum and maximum range for the horizontal axis. In the following demonstration, we constrain the Miles Per Gallon (`mpg`) axis to run exclusively from 15 to 30. As anticipated, any vehicle in the mtcars dataset with an `mpg` value below 15 or above 30 will be excluded from the resulting visualization. This explicit data removal is confirmed by the warning message generated by R.

```
# Create scatterplot with x-axis explicitly ranging from 15 to 30
ggplot(mtcars, aes(mpg, wt)) +
geom_point() +
xlim(15, 30)
```

Warning message:
"Removed 9 rows containing missing values (geom_point)."

The output plot confirms the removal of data points, showing a tighter distribution focused solely on the specified range. The warning message, "Removed 9 rows containing missing values," is the system's way of notifying the user that nine observations were discarded because their X-axis values fell outside the designated boundaries of .

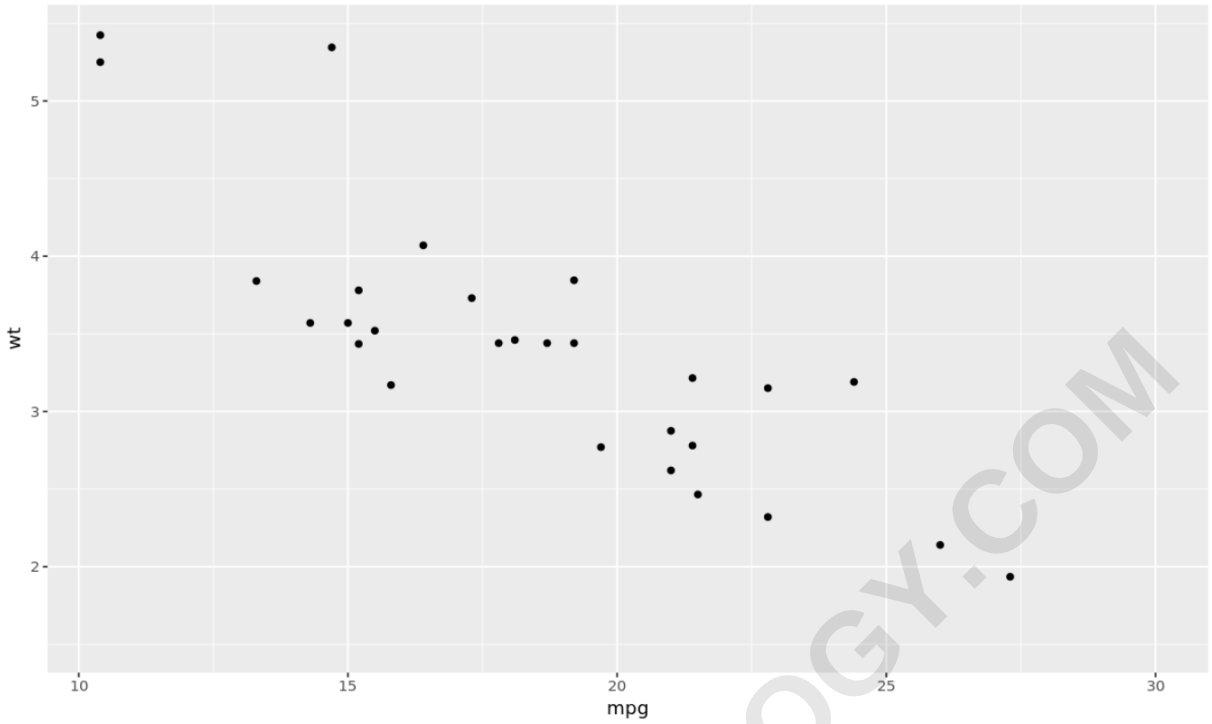


A highly useful feature of both `xlim()` and `ylim()` is the ability to use the special value `NA` (Not Available) to specify that ggplot2 should automatically determine one boundary while the user sets the other. For instance, if we only want to ensure the X-axis does not exceed a certain maximum value (e.g., 30 `mpg`), we can set the lower limit to `NA`. R will then calculate the minimum value based on the available data that remains within the upper constraint.

```
# Create scatterplot setting only the X-axis upper limit at 30
ggplot(mtcars, aes(mpg, wt)) +
geom_point() +
xlim(NA, 30)
```

Warning message:
"Removed 4 rows containing missing values (geom_point)."

In this scenario, four rows were removed because their `mpg` was greater than 30. The lower bound is automatically derived from the minimum `mpg` value of the remaining vehicles. This technique is particularly valuable when ensuring all data points below a certain threshold are included, while clipping any extreme outliers above a defined maximum.



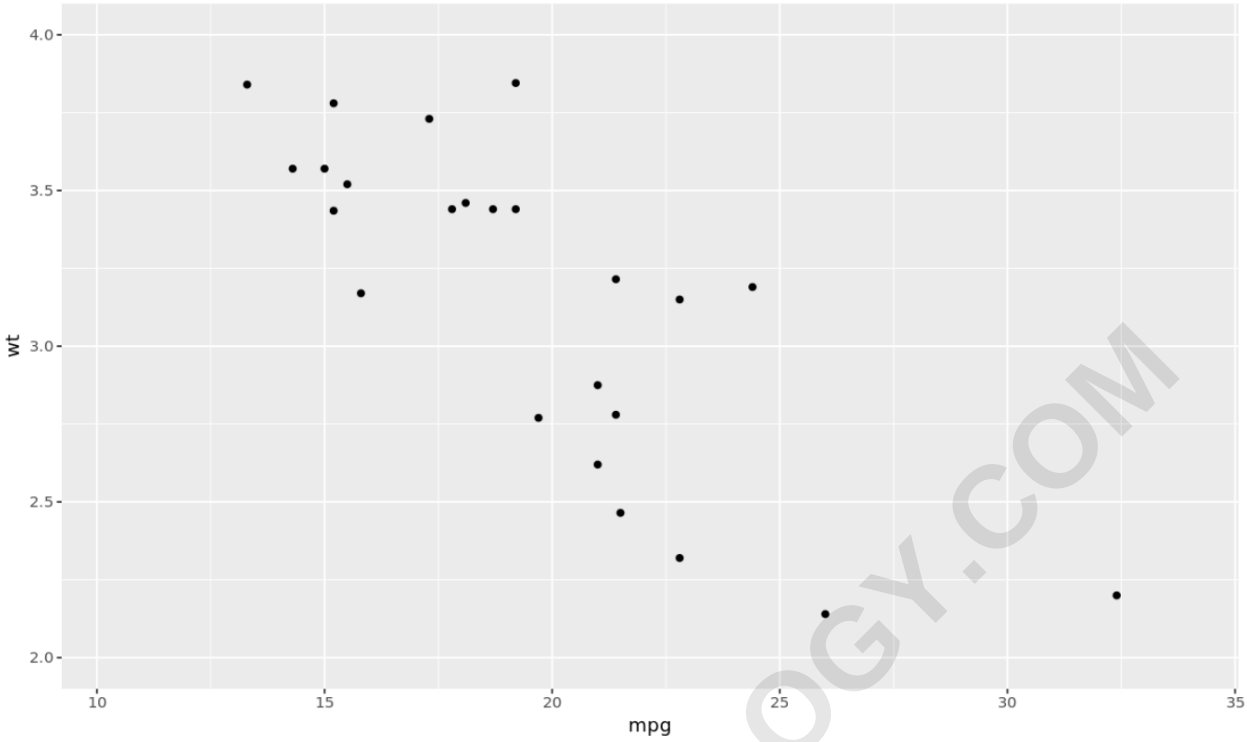
Example 2: Setting Y-Axis Limits Using ylim()

The **`ylim()`** function operates identically to **`xlim()`**, but controls the vertical axis scale, which in our example corresponds to the vehicle weight (**`wt`**). To demonstrate its effect, we will limit the Y-axis range from 2 to 4. This restriction will remove all vehicles weighing less than 2,000 lbs or more than 4,000 lbs from the plotting process, again confirming the function's behavior as a data filter.

```
# Create scatterplot constraining the y-axis (wt) range from 2 to 4
ggplot(mtcars, aes(mpg, wt)) +
geom_point() +
ylim(2, 4)
```

Warning message:
"Removed 8 rows containing missing values (geom_point)."

The resulting graph shows a vertically compressed view of the data, focusing on the mid-range weights. The system reports that 8 rows were removed, confirming that these observations were outside the range and thus excluded entirely from the visual representation. This is a crucial consideration for anyone analyzing the relationship between variables, as omitting data points can significantly alter perceived correlations or trends.

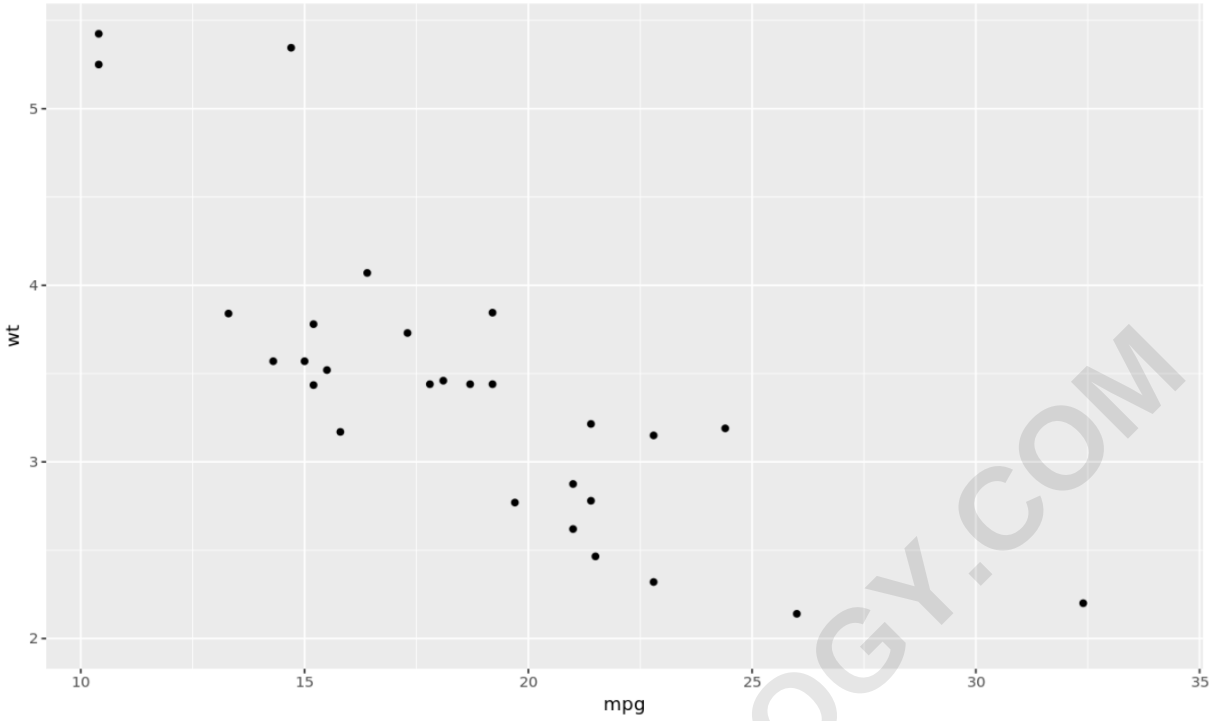


Similar to `xlim()`, `ylim()` supports the use of `NA` for dynamic scaling. Here, we demonstrate setting a lower weight boundary of 2, allowing the upper limit to be determined automatically by the heaviest vehicle remaining in the dataset. This ensures that while we eliminate the lightest vehicles, we retain the full range of weights present above our threshold of 2.

```
# Create scatterplot setting only the Y-axis lower limit at 2
ggplot(mtcars, aes(mpg, wt)) +
geom_point() +
ylim(2, NA)
```

Warning message:
"Removed 4 rows containing missing values (geom_point)."

In this revised example, only 4 observations were removed--specifically, those with a weight less than 2. The upper limit automatically extends to the maximum weight recorded in the dataset, providing a visual perspective that excludes low-weight outliers while keeping the high-end range intact. This flexibility makes `xlim()` and `ylim()` efficient tools for data cleaning and focused preliminary analysis, provided the user accepts the consequence of permanently dropping the excluded observations from the plot calculation.



Example 3: Setting Axis Limits Using coord_cartesian() for Zooming

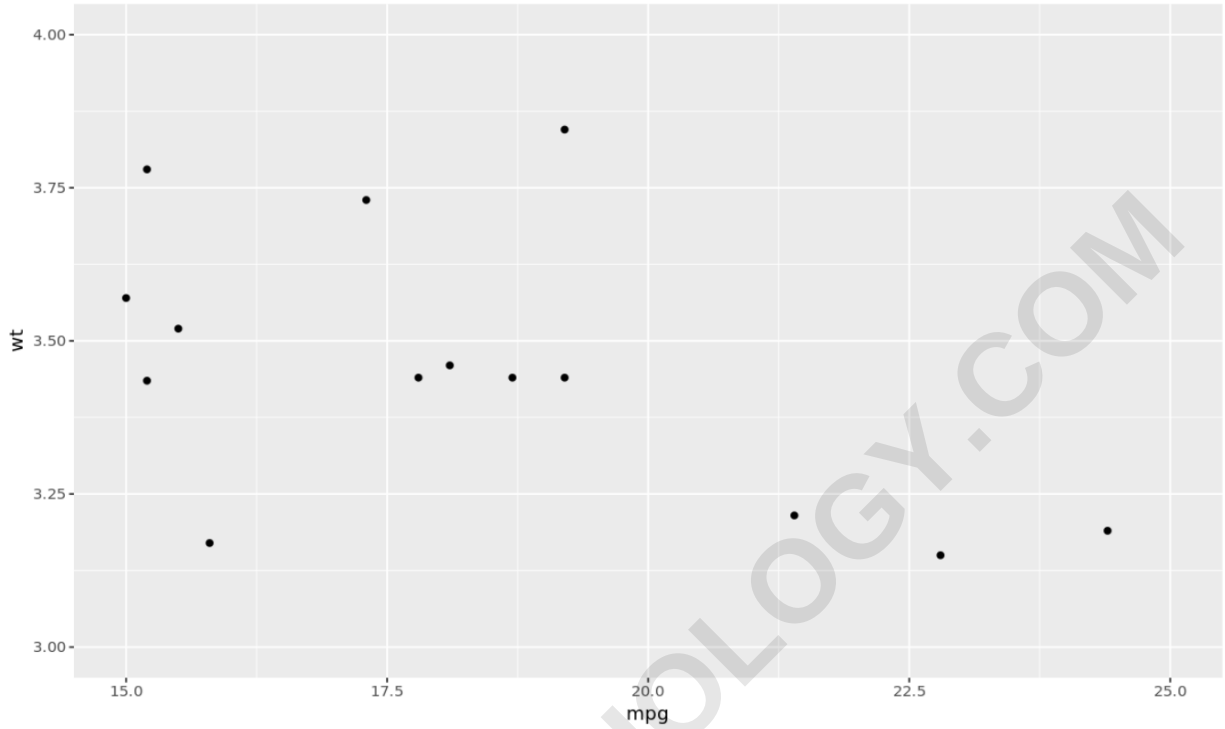
When the requirement is to restrict the visible area of the plot without compromising the underlying data model, **`coord_cartesian()`** is the definitive solution. This function allows for simultaneous control over both the X and Y axes, accepting the desired limits as arguments for **`xlim`** and **`ylim`**. Note that unlike **`xlim()`** and **`ylim()`**, which are standalone functions added as layers, the arguments here are passed directly inside the **`coord_cartesian()`** function call.

In this demonstration, we are setting the X-axis (**`mpg`**) range to and the Y-axis (**`wt`**) range to . Crucially, even though many points fall outside this range, R processes the entire mtcars dataset first, calculates the placement of all points, and only then crops the visual output to the specified coordinate system boundaries.

```
# Create scatterplot, zooming to a specific region using coord_cartesian()
ggplot(mtcars, aes(mpg, wt)) +
geom_point() +
coord_cartesian(xlim = c(15, 25), ylim = c(3, 4))
```

Observe the resulting plot. While the visual boundaries are clearly defined between 15 and 25 on the X-axis and 3 and 4 on the Y-axis, there is an absence of the data removal warning that plagued the previous examples. This silence confirms that no data rows were dropped; they were simply

rendered invisible outside the visual viewport. This makes `coord_cartesian()` the indispensable tool for statistical reporting where the underlying calculations must reflect the full sample size.



Summary of Best Practices for Axis Control

The choice of function in ggplot2 for setting axis limits hinges entirely on the intended outcome: whether the goal is to filter the data or merely to zoom the visualization. Choosing the wrong function can lead to misleading statistical interpretations and unnecessary data loss warnings. Therefore, adopting a clear set of best practices is essential for effective data visualization and analysis.

If your objective is to perform a statistical analysis or visualization exclusively on a subset of the data--for example, analyzing only cars with `mpg` below 30--then utilizing `xlim()` or `ylim()` is appropriate. These functions act as filters, permanently removing unwanted data points from the plot's calculation pipeline. Always be prepared for the warning messages and ensure you document this data exclusion clearly, especially when presenting results.

Conversely, for the vast majority of visual refinement tasks--such as tidying up a plot by removing empty space or focusing on a high-density cluster of points while ensuring trend lines (e.g., smoothing functions) remain accurate across the full range of data--`coord_cartesian()` is the correct and recommended function. It provides the visual control necessary for clarity without performing destructive data manipulation, thereby preserving the statistical integrity of the plot

components.

Mastering these functions allows analysts using R and ggplot2 to produce visualizations that are both aesthetically refined and statistically robust, ensuring that the visual representation aligns perfectly with the underlying analytical intent.

You can find more ggplot2 tutorials here.