How to Easily Split Data into Equal Groups Using R

Authored by

stats writer

November 21, 2025

In the domain of data analysis, the necessity of partitioning a dataset into smaller, coherent subsets is a fundamental requirement for various statistical tests and visualizations. While the standard base R function, split(), is available for dividing data, it typically relies on existing categorical factors. This means the split() function cannot guarantee that the resulting list elements will contain an identical number of observations, especially when dealing with continuous variables.

Achieving truly **equal-sized groups**--where the count of observations in each subset is consistent--is critical for analytical techniques such as creating balanced bins for regression analysis or generating stratified samples. For continuous numeric data, this process requires techniques rooted in **quantiles**, ensuring that regardless of the variable's distribution shape, the resulting bins contain an equal frequency of data points. This article demonstrates how to use the specialized function within the ggplot2 package to perform this precise data partitioning in R.

Introducing the `cut_number()` Function for Quantile Binning

To reliably segment a numeric vector into equal-sized buckets, the most effective tool is the **cut_number()** function, which is provided as part of the powerful ggplot2 package. Unlike base R's standard `cut()` function, which defines bins based on breaks in the variable's value (equal width), **cut_number()** creates bins based on the distribution of observations, ensuring that each bin contains approximately the same number of data points. This technique is often referred to as **quantile** binning or equal-frequency discretization.

The **cut_number()** function automatically calculates the appropriate **quantiles** of the input data and uses these values as the boundaries for the groups. For instance, if you request four groups (quartiles), the function determines the values corresponding to the 25th, 50th, and 75th percentiles, and then uses these values to define the four resultant groups. This methodology is critical when the underlying data distribution is highly skewed, as it prevents the issue of having a vast majority of observations clustered into one or two bins while leaving others sparse.

By relying on quantiles, **cut_number()** provides a robust mechanism for creating balanced subgroups. The input must be a continuous variable, and the resulting output is a factor variable specifying the interval range for each observation.

Understanding the Syntax and Required Arguments

The structure of the **cut_number()** function is straightforward, requiring only the input data and the desired number of partitions. Proper implementation requires loading the ggplot2 package first, as this function is not part of the standard base R distribution.

This function uses the following basic syntax:

cut_number(x, n)

where:

x: Name of **numeric vector** to split. This parameter specifies the column from a data frame that you wish to partition.
n: The integer value representing the precise **number of groups** that the input data (x) should be divided into. Setting n=10 creates deciles.

The following example shows how to use this function in practice, demonstrating both the setup of the dataset and the application of the function itself.

Example Setup: Creating the Sample Dataset

Suppose we are analyzing performance statistics for a basketball team and have recorded the total points scored by 12 different players. We want to divide these 12 players into three groups of equal size (four players each) based on their scoring performance.

We will create a data frame in R containing player identifiers and their respective scores. This setup provides a clear, reproducible scenario for demonstrating quantile grouping.

#create data frame
df <- data.frame(player=LETTERS,
points=c(1, 2, 2, 2, 4, 5, 7, 9, 12, 14, 15, 22))

#view data frame
df

player points
1 A 1
2 B 2
3 C 2
4 D 2
5 E 4
6 F 5
7 G 7
8 H 9
9 I 12
10 J 14
11 K 15
12 L 22

How to Use LETTERS Function in R

Implementing Equal Grouping Using `cut_number()`

We can use the **cut_number()** function to create a new column called `group` that splits each row in the data frame into one of three groups based on the value in the `points` column. Since N=12 and we request n=3 groups, each group is guaranteed to contain 4 players.

We must first load the ggplot2 library to access this specialized binning function. The command below performs the binning and assigns the factor output directly to the new column.

library(ggplot2)

#create new column that splits data into three equal sized groups based on points
df$group <- cut_number(df$points, 3)

#view updated data frame
df

player points group
1 A 1
2 B 2
3 C 2
4 D 2
5 E 4 (3.33,10]
6 F 5 (3.33,10]
7 G 7 (3.33,10]
8 H 9 (3.33,10]
9 I 12 (10,22]
10 J 14 (10,22]
11 K 15 (10,22]
12 L 22 (10,22]

Each of the 12 players has been successfully placed into one of three groups based on the value in the `points` column. The output confirms that four players are assigned to each interval, validating the equal-frequency binning process.

Interpreting the Factor Levels (Interval Ranges)

The resulting groups are defined by the calculated quantile breaks of the `points` data. These factor levels represent specific, non-uniform interval ranges:

group 1: . This group includes players with scores from 1 up to 3.33, covering the lowest four observations.
group 2: `(3.33, 10]`. This group captures the middle segment of scores, strictly greater than 3.33 up to 10.
group 3: `(10, 22]`. This final group encompasses the highest scores, strictly above 10 up to the maximum score of 22.

We can clearly see that the interval widths are not equal (e.g., the first group spans 2.33 points, while the last group spans 12 points). This disparity in interval size is a direct consequence of prioritizing observation count over value range--a hallmark of quantile binning.

Converting Group Labels to Integer Values

For simplified downstream analysis or modeling, it is often preferred to display the group membership as sequential integers (1, 2, 3) rather than interval notation. Since the output of **cut_number()** is an ordered factor, we can use the **as.numeric()** function to easily convert these levels into their corresponding numerical index.

We achieve this by wrapping the **cut_number()** function call within the **as.numeric()** function. This transformation assigns the integer 1 to the first factor level, 2 to the second, and so on, based on the inherent order of the quantile breaks.

library(ggplot2)

#create new column that splits data into three equal sized groups based on points
df$group <- as.numeric(cut_number(df$points, 3))

#view updated data frame
df

player points group
1 A 1 1
2 B 2 1
3 C 2 1
4 D 2 1
5 E 4 2
6 F 5 2
7 G 7 2
8 H 9 2
9 I 12 3
10 J 14 3

11 K 15 3
12 L 22 3

The new `group` column now contains the values 1, 2, and 3 to indicate which group the player belongs to. Once again, each group consistently contains four players, demonstrating that the use of **as.numeric()** only alters the label format, not the underlying group assignment defined by the quantiles.

Conclusion and Practical Considerations

The **cut_number()** function provides an indispensable, robust method for partitioning continuous data into subsets of equal size within R. This technique is particularly useful in fields requiring balanced comparisons, such as A/B testing analysis, demographic stratification, or performance benchmarking.

It is important to reiterate that while **cut_number()** strives for absolute equality, significant ties in the input **numeric vector** near a calculated quantile break may lead to minor deviations in group sizes. However, for most datasets, the resulting bins will be highly balanced.

Note: To split the points column into more than three groups, simply change the `3` in the `cut_number()` function to a different integer (e.g., 4 for quartiles or 10 for deciles). The total number of observations must be divisible by or reasonably close to the number of groups requested for optimal results.