how to use createDataPartition() Function in R

Authored by

stats writer

November 22, 2025

In the field of data science and machine learning, successfully evaluating a predictive **model** depends critically on how the input **dataset** is divided. A standard practice involves segmenting the data into distinct subsets, typically referred to as **training and test sets**. This essential segmentation process ensures that the model is trained on one subset of data and evaluated impartially on unseen data, providing a true measure of generalization ability.

The **createDataPartition()** function, residing within the highly utilized **caret package** in the **R** programming environment, is designed specifically for this purpose. It offers a powerful, efficient, and balanced method for splitting a data structure, such as a **data frame**, ensuring that the critical outcome variable is represented proportionally across the resulting partitions. This technique is known as stratified sampling.

Understanding Stratified Data Partitioning in R

To properly initiate a robust machine learning workflow in **R**, the **createDataPartition()** function is indispensable. Unlike simple random sampling which might unintentionally skew the distribution of the target variable between partitions, **createDataPartition()** performs stratification based on the outcome variable specified. This is especially vital when dealing with unbalanced datasets or ensuring consistent class representation, thereby enhancing the reliability of subsequent **model building** and evaluation.

The primary goal of using this function is to generate indices corresponding to the observations that should be included in the designated training set, allowing for seamless subsetting of the original **data frame** into the required partitions (training and testing).

Core Syntax and Essential Arguments

The **createDataPartition()** function follows a precise structure, requiring key arguments to define the partitioning process. Understanding these parameters is crucial for controlling the size and composition of the resulting subsets.

The function utilizes the following basic syntax structure:

createDataPartition(y, times = 1, p = 0.5, list = TRUE, ...)

The essential components are defined as follows:

y: This is the vector representing the outcome variable (or the factor/class variable) upon which the stratification should occur. This is the variable whose distribution must be preserved across subsets.
times: Specifies the number of separate partitions or data splits to generate. For standard

training/testing split, this value is usually set to 1.
p: This argument defines the proportion (percentage) of the observations that should be allocated to the initial training set. For instance, a value of 0.8 allocates 80% of the data to training.
list: A logical value determining the output format. If **TRUE** (default), results are stored in a list. Setting this to **FALSE** returns a single matrix or vector of indices, which is often easier for direct indexing operations.

Practical Application: Setting up the Sample Data

To illustrate the power and utility of **createDataPartition()**, we will work through a concrete example. Suppose we are analyzing a simulated data structure in R containing 1,000 observations. This **dataset** tracks student performance, specifically correlating the total **hours** studied with their corresponding final examination **score**.

The initial step involves creating this sample data frame and examining its structure using R code:

 # Ensure reproducibility of the random sampling process
set.seed(0)

# Construct the sample data frame with 1000 rows
df <- data.frame(hours=runif(1000, min=0, max=10),
score=runif(1000, min=40, max=100))

# Display the first few observations to verify structure
head(df)

hours score
1 8.966972 55.93220
2 2.655087 71.84853
3 3.721239 81.09165
4 5.728534 62.99700
5 9.082078 97.29928
6 2.016819 47.10139

Defining the Training and Testing Objective

Our objective is to develop a predictive model that utilizes the 'hours' variable to accurately forecast the final exam 'score'. To rigorously assess the performance of this model, we must adhere to the best practices of partitioning the data.

We define a common split ratio for this exercise: we desire to train the predictive model using 80%

of the observations in the **dataset**, reserving the remaining 20% for testing the model's performance on previously unseen data points. This ratio (80/20) is fundamental for evaluating generalization ability.

Implementing Data Splitting with createDataPartition()

To execute the desired 80/20 split while maintaining proportionality based on the 'score' variable, we invoke the **createDataPartition()** function. We must first load the **caret package**. We set the proportion 'p' to 0.8 and, critically, set 'list' to **FALSE** so that the output is a vector of row indices, facilitating straightforward subsetting. The stratification is based on the target variable, `df$score`.

The following sequence of commands demonstrates how to use the returned indices to construct the final training and testing data frames:

library(caret)

# Generate indices for the 80% training set, stratified by score
train_indices <- createDataPartition(df$score, times=1, p=.8, list=FALSE)

# Create the training set using the generated indices
df_train <- df

# Create the testing set using the negation of the indices (the remaining 20%)
df_test <- df

Verifying Partition Integrity and Size

After the execution of the partitioning code, it is essential to verify that the split resulted in the expected number of rows for both the **training and test sets**. Since the original **data frame** contained 1,000 observations, an 80% split should yield 800 rows for training and 200 rows for testing.

We use the `nrow()` function to confirm these counts:

 # Check the size of the training set
nrow(df_train)

800

# Check the size of the testing set
nrow(df_test)

200

The confirmation shows that our training dataset contains 800 rows, accounting for exactly 80% of the total observations. Similarly, the test dataset holds the remaining 200 rows, validating the partition process performed by **createDataPartition()**.

Examining the Subsetted Data Frames

Finally, we can visually inspect the first few rows of both the training and testing data frames to ensure that the observations have been randomly, yet stratified, distributed between the two sets. Note that the row names (the original indices) confirm which observations were allocated to which set.

 # View the head of the training set
head(df_train)

hours score
1 8.966972 55.93220
2 2.655087 71.84853
3 3.721239 81.09165
4 5.728534 62.99700
5 9.082078 97.29928
7 8.983897 42.34600

# View the head of the testing set
head(df_test)

hours score
6 2.016819 47.10139
12 2.059746 96.67170
18 7.176185 92.61150
23 2.121425 89.17611
24 6.516738 50.47970
25 1.255551 90.58483

Conclusion and Next Steps

The use of **createDataPartition()** from the **caret package** provides a significant advantage over manual or non-stratified splitting methods. By ensuring that the distribution of the key outcome variable is preserved in both the training and testing partitions, it minimizes sampling bias and

maximizes the chance of generating a predictive model that generalizes effectively to new data.

This systematic approach to creating robust data partitions is fundamental to accurate statistical analysis and model validation in **R**. With the data now properly segmented, the workflow can proceed to the modeling phase, utilizing the training set (`df_train`) to fit the algorithm and subsequently evaluating its performance using the unseen test set (`df_test`).