How to Calculate Quantiles in R Using the quantile() Function

Authored by

stats writer

December 4, 2025

The `quantile()` function is one of the most fundamental and powerful tools available in R for conducting exploratory data analysis and generating descriptive summaries. At its core, this function is designed to calculate the quantiles for a specified set of data, effectively segmenting the distribution into equal, ordered proportions. Key outputs derived from this function automatically include essential summary statistics such as the **minimum** (0th quantile), **maximum** (100th quantile), the **median** (50th percentile), and the common spread indicators, which are the **first quartile** (25th percentile) and the **third quartile** (75th percentile).

Beyond these standard measurements, the flexibility of the `quantile()` function allows users to calculate any arbitrary quantile or percentile, providing a granular view of data distribution. Utilizing this function is crucial for analysts seeking a deeper understanding of the central tendency, skewness, and overall spread of a data set. By efficiently generating these descriptive statistics, `quantile()` facilitates robust preliminary analysis before moving onto more complex statistical modeling.

Introduction to Quantiles and Descriptive Statistics

In the realm of descriptive statistics, **quantiles** represent values that partition a sorted, ranked dataset into equal-sized subgroups. These partitioning points are essential because they provide context regarding the placement of specific data points relative to the overall distribution. For instance, knowing the median (the 0.5 quantile) tells us the value below which 50% of the data falls. Depending on the number of divisions, these quantiles are given specific names: quartiles divide the data into four parts, quintiles into five, and deciles into ten. Understanding these divisions is vital for identifying outliers, assessing data symmetry, and visualizing distribution spread through tools like box plots.

The implementation of this statistical concept in R is handled primarily by the **`quantile()`** function, which calculates the sample quantiles directly from an input dataset. Unlike simple summary functions that only return measures of central tendency, `quantile()` offers a comprehensive overview of how the data values are dispersed across their range. This makes it an indispensable tool during the initial stages of any data analysis pipeline, allowing researchers to quickly gauge the dispersion and concentration of observations across the dataset.

Syntax and Core Arguments of `quantile()`

To effectively utilize this powerful tool, it is important to understand its core structure and parameters. The primary usage of the `quantile()` function in R adheres to a straightforward structure, ensuring ease of application across various data formats. The function is designed to be highly customizable, allowing the user to specify exactly which partitioning points they wish to calculate, moving far beyond the standard quartiles if necessary.

This function uses the following basic syntax, which clearly defines the input data, the desired probabilities, and how missing values should be handled:

quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE)

The arguments governing the function's behavior are crucial for accurate computation:

x: This required argument is the name of the input vector or numeric data array from which the quantiles are to be calculated. It must contain numeric data.
probs: This argument is a numeric vector specifying the probability levels at which quantiles should be determined. By default, it uses `seq(0, 1, 0.25)`, which automatically calculates the 0%, 25%, 50%, 75%, and 100% quantiles (i.e., the five-number summary). Custom probabilities must range between 0 and 1.
na.rm: This logical argument dictates how the function handles missing values (NA). If set to `TRUE`, any missing values in the input data `x` are removed before quantile calculation; if set to `FALSE` (the default), the function will return `NA` if any missing values are present.

The following practical examples demonstrate how to implement this function across different scenarios, including simple vectors, columns within a data frame, and grouped analyses.

Example 1: Calculating Quantiles for a Simple Vector

When working with a single set of numeric observations, the `quantile()` function is straightforward to apply. This scenario is common when analyzing survey results, measurement errors, or any unidimensional dataset. The primary goal is often to calculate the standard quartiles to determine the interquartile range (IQR), which measures the statistical dispersion and robustness against outliers.

The following code illustrates how to define a numeric vector of data points and then calculate various quantiles, demonstrating the function's versatility in customizing the divisions of the data distribution. Notice how changing the step size in the `seq()` function within the `probs` argument alters the type of quantile calculated.

#define vector of data
data = c(1, 3, 3, 4, 5, 7, 8, 9, 12, 13, 13, 15, 18, 20, 22, 23, 24, 28)

#calculate quartiles (dividing data into four equal parts: 0, 0.25, 0.50, 0.75, 1.0)
quantile(data, probs = seq(0, 1, 1/4))

0% 25% 50% 75% 100%
1.0 5.5 12.5 19.5 28.0

```
#calculate quintiles (dividing data into five equal parts: 0, 0.20, 0.40, 0.60, 0.80, 1.0)
quantile(data, probs = seq(0, 1, 1/5))

0% 20% 40% 60% 80% 100%
1.0 4.4 8.8 13.4 21.2 28.0

#calculate deciles (dividing data into ten equal parts: 0.1 increments)
quantile(data, probs = seq(0, 1, 1/10))

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
1.0 3.0 4.4 7.1 8.8 12.5 13.4 17.7 21.2 23.3 28.0

#calculate random quantiles of interest (e.g., the 20th, 50th, and 90th percentiles)
quantile(data, probs = c(.2, .5, .9))

20% 50% 90%
4.4 12.5 23.3
```

Calculating Different Types of Quantiles (Quartiles, Deciles, Percentiles)

The strength of the `quantile()` function lies in its ability to adapt to different requirements for distributional division. When `probs` is set to `seq(0, 1, 0.25)`, we calculate **quartiles**, which are perhaps the most frequently used statistical quantiles. The resulting values--Q1 (25%), Q2 (Median, 50%), and Q3 (75%)--help define the box in a boxplot visualization and are key components of the five-number summary used in descriptive statistics. The interquartile range (IQR = Q3 - Q1) provides a robust measure of data spread, less sensitive to extreme outliers than the standard deviation.

When a finer granularity is needed, such as in economic studies or demographic analyses, the function can be configured to calculate **deciles** (10 equal parts) or **quintiles** (5 equal parts), as demonstrated in the previous code block. For deciles, the `probs` argument is defined using a step size of 0.1 (or 1/10). Deciles are particularly useful when segmenting a population or variable based on tenths, such as income distribution, providing a more detailed look into where the majority of observations lie and identifying significant concentration points within the dataset.

Ultimately, the concept of a **percentile** is simply a quantile expressed as a percentage. By providing a custom vector of probabilities to the `probs` argument--for example, `c(0.01, 0.99)`--an analyst can precisely determine the values that define the bottom 1% and the top 1% of the distribution. This ability to target specific points within the distribution is invaluable for tasks such as setting performance benchmarks, defining safety thresholds, or identifying extreme values that require further investigation.

Example 2: Analyzing Quantiles in R Data Frames

In real-world data analysis, datasets are usually structured as data frames, where data is organized into rows (observations) and columns (variables). To calculate quantiles for a specific variable within a data frame, the column must be extracted using the `$` operator or bracket notation before being passed to the `quantile()` function. This ensures that the function operates only on the target numeric vector, adhering to the standard input requirement.

The following code demonstrates the creation of a sample data frame and the subsequent calculation of quartiles specifically for the column labeled 'var2'. This method is highly effective when the analyst is focused on profiling the distribution of individual variables independently of others within the dataset.

```
#create data frame
df <- data.frame(var1=c(1, 3, 3, 4, 5, 7, 7, 8, 12, 14, 18),
var2=c(7, 7, 8, 3, 2, 6, 8, 9, 11, 11, 16),
var3=c(3, 3, 6, 6, 8, 4, 4, 7, 10, 10, 11))

#calculate quartiles of column 'var2'
quantile(df$var2, probs = seq(0, 1, 1/4))

0% 25% 50% 75% 100%
2.0 6.5 8.0 10.0 16.0
```

A more efficient approach for computing summary statistics across multiple columns simultaneously involves the use of the `sapply()` or `lapply()` functions. The `sapply()` function applies the `quantile()` calculation iteratively to every specified column in the data frame, streamlining the process of generating descriptive summaries for the entire dataset. This is particularly valuable for data auditing and high-level summaries.

```
#calculate quartiles of every column
sapply(df, function(x) quantile(x, probs = seq(0, 1, 1/4)))

var1 var2 var3
0% 1.0 2.0 3
25% 3.5 6.5 4
50% 7.0 8.0 6
75% 10.0 10.0 9
100% 18.0 16.0 11
```

The output matrix clearly shows the five-number summary for each variable (var1, var2, var3), providing a comparative snapshot of the distributional characteristics of each column. For example, 'var1' has a median of 7.0, while 'var2' has a median of 8.0, indicating slightly different central tendencies across the measured features.

Example 3: Grouped Quantile Calculation using `dplyr`

Modern data analysis frequently requires calculating statistics not just across an entire dataset, but separately for different subgroups defined by categorical variables. This type of conditional calculation is best performed using functions from the powerful `dplyr` package, which is part of the Tidyverse ecosystem in R. By combining the `group_by()` and `summarize()` verbs, we can apply the `quantile()` function to specific subsets of data.

The process involves first loading the `dplyr` library and then piping the data frame through the necessary steps. The `group_by()` function organizes the data according to the categorical variable (e.g., 'team'), and the `summarize()` function then computes the desired quantiles for a numeric variable (e.g., 'points') within each of those defined groups. This approach provides incredibly detailed and context-specific descriptive statistics.

library(dplyr)

#define data frame with a grouping variable ('team')
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'C', 'C', 'C'),
points=c(1, 3, 3, 4, 5, 7, 7, 8, 12, 14, 18))

#define quantiles of interest (25th, 50th, 75th percentiles)
q = c(.25, .5, .75)

#calculate quantiles by grouping variable using dplyr workflow
df %>%
group_by(team) %>%
summarize(quant25 = quantile(points, probs = q),
quant50 = quantile(points, probs = q),
quant75 = quantile(points, probs = q))

A tibble: 3 x 4
team quant25 quant50 quant75

1 A 2.5 3 3.25
2 B 6.5 7 7.25
3 C 13 14 16

The resulting tibble clearly shows the quantile differences between the teams. For instance, Team A has a median (quant50) of 3, indicating a low score concentration, while Team C has a much higher median of 14. This grouped analysis is far more insightful than calculating the quantiles for the 'points' column across all teams combined, as it correctly identifies the heterogeneity in the data distribution attributable to the grouping variable.

Advanced Considerations: Handling Missing Data and Type Arguments

While the core usage of `quantile()` is straightforward, advanced users must consider two important aspects: the treatment of missing data and the method used for quantile estimation. As noted in the syntax definition, the `na.rm` argument controls how `NA` (Not Available) values are handled. Failing to set `na.rm = TRUE` when missing data are present will result in the function returning `NA` for all quantile calculations, stopping the analysis prematurely. Ensuring proper handling of missing data is a critical step in maintaining the integrity of the quantile estimation.

Furthermore, `quantile()` includes an optional argument called `type`, which specifies one of the nine different algorithms (or methods) R uses for calculating sample quantiles. These methods differ primarily in how they handle interpolation when the desired quantile falls between two actual data points. Method 7, the default, is widely used, but methods 8 and 9 (which estimate quantiles suitable for statistical modeling) are also common. While most standard analyses can rely on the default method, statistical rigor may necessitate testing different types, especially with smaller or heavily skewed datasets, as the choice of type can slightly influence the resulting quantile estimates, particularly for extreme percentiles.

In summary, the `quantile()` function provides robust and flexible functionality for distributional analysis in R. Whether calculating simple quartiles for a vector, summarizing columns in a data frame, or performing complex grouped analysis using tools like `dplyr`, this function is foundational to exploratory data analysis.

Further Learning Resources

For analysts seeking to expand their knowledge of distributional metrics, the following tutorials show how to use the **`quantile()`** function to calculate other common quantile values and related statistical outputs: