How to Easily Calculate Quantiles by Group in R

Authored by

stats writer

December 6, 2025

In the field of **statistics** and **data science**, understanding the distribution of data is paramount. Measures of central tendency, such as the mean, offer a limited view, especially when distributions are skewed or contain outliers. This is where **quantiles** become indispensable. Simply put, quantiles are values that meticulously divide a ranked dataset into equal, contiguous subgroups. The most common examples are quartiles, which divide the data into four equal parts, or percentiles, which divide it into 100 parts. These measures help analysts gain a robust understanding of the spread and concentration points of numerical variables.

The calculation of overall quantiles for an entire dataset is straightforward. However, real-world data is rarely monolithic; it usually contains categorical variables that define meaningful subpopulations. To derive actionable insights, it is often necessary to perform a **grouped analysis**, calculating these descriptive statistics not for the whole population, but separately for each distinct sub-group defined by a categorical variable. For instance, in a sales dataset, one might need the median sales figure per region, or the 75th percentile of employee salaries per department. This detailed, segmented approach allows practitioners to identify internal disparities, benchmarks, and performance differences across categories.

This tutorial focuses on achieving this specific task efficiently using **R**, the leading environment for statistical computing. While base R provides functions for calculating quantiles, the process of applying this calculation across specified groups is significantly streamlined by leveraging the capabilities of the **dplyr** package, which is part of the tidyverse ecosystem. By combining the power of grouping with summarization functions, we can generate clean, informative reports that segment complex numerical distributions based on categorical factors, providing clarity and depth to any data analysis project.

General Syntax for Grouped Quantile Calculation

To accurately calculate quantiles segmented by a categorical variable, we rely heavily on the architecture provided by the **dplyr** package. This package introduces a syntax known as pipelining (using the %>% operator), which enhances readability and allows complex multi-step operations to be chained together logically. The core process involves three critical steps: loading the necessary library, defining the specific quantiles of interest, and applying the grouping and summarization pipeline.

The fundamental approach utilizes the `group_by()` function to partition the dataset based on a chosen categorical variable, followed immediately by the `summarize()` function, which executes the calculation on each subgroup independently. Within `summarize()`, we use the base R function `quantile()`. The `quantile()` function requires two primary arguments: the numerical variable upon which the calculation is performed, and the `probs` argument, which is a vector defining the probabilities (or desired quantiles) ranging from 0 to 1. By assigning the results of `quantile()` to

new, descriptive column names, we generate a final output table where each row represents a unique group and its corresponding calculated **quantiles**.

The following structure outlines the general code pattern required to execute this powerful grouped calculation. It uses placeholders for the dataset (`df`), the grouping variable (`grouping_variable`), and the variable being measured (`numeric_variable`). This template serves as the foundation for all subsequent examples, ensuring that the calculation of the 25th, 50th (median), and 75th percentiles (quartiles) is performed efficiently across all groups.

library(dplyr)

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

#calculate quantiles by grouping variable
df %>%
group_by(grouping_variable) %>%
summarize(quant25 = quantile(numeric_variable, probs = q),
quant50 = quantile(numeric_variable, probs = q),
quant75 = quantile(numeric_variable, probs = q))

Understanding this structure is key to mastering grouped data manipulation in **R**. The use of the `group_by()` function ensures that all subsequent operations, specifically the `summarize()` call, are applied contextually to subsets of the data. This provides a clear, aggregated summary table detailing the distribution metrics for each category, which is often far more insightful than analyzing the raw data itself.

Practical Example 1: Calculating Quartiles for Sample Data

To demonstrate the utility of grouped quantile calculation, let us create a sample dataset focused on sports performance. We will generate a data frame containing information about various teams (Team A, B, and C) and the number of wins achieved by each team over multiple seasons or periods. Our goal is to calculate the quartiles--specifically the 25th, 50th (median), and 75th percentiles--for the number of wins, grouped separately for each team. This analysis will instantly reveal the typical performance range and the distribution skewness unique to each team, allowing for quick comparisons of performance consistency.

The following code block first constructs the sample data frame, `df`, ensuring a balanced distribution of observations across the three categorical teams. We then load the necessary **dplyr** library, which is essential for executing the grouped operations. Following the data creation, a preliminary view of the data structure (using `head(df)`) helps confirm that the data is correctly

structured with the categorical variable `team` and the numerical variable `wins`.

The critical step involves defining the probabilities vector `q` as `c(.25, .5, .75)`, corresponding to the desired quartiles. We then pipeline the dataset, first grouping by the `team` variable, and subsequently summarizing the results. The `summarize()` function calculates the respective quantiles for the `wins` column within each team subset. Note how we explicitly index the `q` vector (e.g., `q` for the 25th percentile) to ensure clarity in the output column assignment.

library(dplyr)

#create data
df <- data.frame(team=c('A', 'A', 'A', 'A', 'A', 'A', 'A', 'A',
'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B',
'C', 'C', 'C', 'C', 'C', 'C', 'C', 'C'),
wins=c(2, 4, 4, 5, 7, 9, 13, 13, 15, 15, 14, 13,
11, 9, 9, 8, 8, 16, 19, 21, 24, 20, 19, 18))

#view first six rows of data
head(df)

team wins
1 A 2
2 A 4
3 A 4
4 A 5
5 A 7
6 A 9

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

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

team quant25 quant50 quant75
1 A 4 6 10
2 B 9 12 14.2
3 C 17.5 19 20.2

The resulting output clearly shows the quartile distribution for each team. For instance, Team A has a median (50th **quantile**) of 6 wins, indicating that half of its recorded seasons resulted in 6 wins or fewer. Team C, conversely, exhibits much stronger performance, with its 25th percentile (17.5 wins) already exceeding Team B's median (12 wins). This example powerfully demonstrates how grouped **quantiles** provide immediate, comparative insights into distributional characteristics that simple means might obscure, making it a staple technique in exploratory **data science**.

Expanding the Analysis: Specifying Custom Quantiles

While the calculation of quartiles (25th, 50th, 75th percentiles) is standard, statistical analysis often requires more granular or specific percentile divisions to understand certain thresholds or cut-off points within the data. For example, an analyst might be interested in the 20th, 40th, 60th, and 80th percentiles to divide the data into five equal groups (quintiles), providing a finer resolution of the distribution compared to quartiles.

The flexibility of the **R** `quantile()` function allows us to define any arbitrary set of probabilities for calculation. The only necessary modification to our established `group_by()` and `summarize()` structure is redefining the vector of probabilities, `q`. Instead of `c(.25, .5, .75)`, we can substitute any desired sequence of values between 0 and 1.

In the following demonstration, we redefine the `q` vector to calculate the quintiles, thereby partitioning the data into five distinct segments. Crucially, the subsequent `summarize()` call must be updated to match the number of quantiles defined in `q`, ensuring that each calculated percentile is assigned a meaningful output column name (e.g., `quant20`, `quant40`, etc.). This adaptability ensures that the analysis is not limited to conventional statistical measures but can be tailored precisely to the research question at hand.

#define quantiles of interest
q = c(.2, .4, .6, .8)

#calculate quantiles by grouping variable
df %>%
group_by(team) %>%
summarize(quant20 = quantile(wins, probs = q),
quant40 = quantile(wins, probs = q),
quant60 = quantile(wins, probs = q),
quant80 = quantile(wins, probs = q))

team quant20 quant40 quant60 quant80

1 A 4 4.8 7.4 11.4

2 B 9 10.6 13.2 14.6
3 C 16.8 18.8 19.2 20.6

Observing the output reveals a much finer breakdown of performance. For Team A, the difference between the 20th percentile (4 wins) and the 40th percentile (4.8 wins) is very small, indicating a tight clustering of lower performance results. Conversely, the spread between the 60th and 80th percentiles (7.4 to 11.4) is wider, suggesting a more varied distribution in the upper half of their performance records. This ability to define and calculate custom **quantiles** by group is a powerful feature for analysts looking to thoroughly characterize heterogeneous data distributions.

Advanced Technique: Calculating a Single Arbitrary Percentile

Sometimes, the analytical focus is extremely narrow, concentrating on a single, high-stakes threshold rather than the full distribution. A common request is to calculate a specific, high percentile, such as the 90th or 95th, which represents the performance level achieved by only the top 10% or 5% of observations within each group. This is particularly useful for setting high performance benchmarks, identifying top performers, or defining extreme value criteria in fields like finance or quality control.

Calculating just one arbitrary percentile is the simplest variation of the grouped quantile approach. Instead of defining a vector `q` and indexing its elements, we can directly input the desired probability value into the `probs` argument of the `quantile()` function within the `summarize()` command. This eliminates the need for creating and managing an external vector, simplifying the code block considerably.

For example, if we want to determine the 90th percentile of wins for each team, we set `probs = 0.9`. This calculation informs us of the minimum number of wins required to be considered in the top 10% of performance for that specific team. The structure remains focused on using `group_by()` to maintain segregation of the teams, followed by a succinct `summarize()` call that executes the specific percentile calculation.

#calculate 90th percentile of wins by team
df %>%
group_by(team) %>%
summarize(quant90 = quantile(wins, probs = 0.9))

team quant90

1 A 13
2 B 15
3 C 21.9

The result provides clear benchmarks. For Team A, achieving 13 wins places a season in the top 10% of their historical performance, while for the high-performing Team C, 21.9 wins is the required threshold. This single-value calculation is crucial when the objective of the **grouped analysis** is primarily focused on identifying outliers, setting targets, or performing non-parametric checks on upper limits. This method confirms the flexibility of the **R** environment combined with the efficient data manipulation capabilities of the **dplyr** package.

Considerations for Quantile Calculation in R

While the `quantile()` function appears straightforward, practitioners must be aware of its underlying methodology, particularly the different algorithms used for interpolation. The base R `quantile()` function offers nine distinct types of algorithms (specified by the `type` argument) for calculating quantiles, which primarily differ in how they handle interpolation when the desired **quantile** position falls between two observed data points. The default setting is `type=7`, which is the standard choice recommended by Hyndman and Fan for robust statistical analysis, but the choice can significantly impact results, especially in small datasets.

For large datasets, the choice of `type` usually has minimal practical impact. However, in scenarios where precision is critical or when comparing results with external software (which might use a different default algorithm, such as `type=6` used by SAS or Minitab), explicitly specifying the `type` argument within the `quantile()` function call is essential for reproducibility and consistency. For example, if we wanted to enforce the Type 8 algorithm for a specific analysis, the code snippet within the `summarize()` function would look like `quantile(numeric_variable, probs = q, type = 8)`.

Furthermore, grouped quantile calculation inherently handles missing values (`NA`s) by default, meaning they are ignored in the calculation unless otherwise specified. If retaining `NA` values in the input variable should result in an `NA` quantile output for that group, the user would need to ensure the `na.rm = FALSE` argument is explicitly passed to the `quantile()` function, although the default setting `na.rm = TRUE` (remove missing values) is usually preferred in **grouped analysis** to ensure robust calculation based only on available data points within each group.

Conclusion: Summarizing Key Takeaways

Mastering the calculation of **quantiles** by group is a fundamental skill for any analyst working in **R**, providing a powerful means to understand complex, segmented data distributions. By combining the data wrangling efficiency of the **dplyr** package, particularly the `group_by()` function, with the statistical depth of the base R `quantile()` function, we can generate highly informative summaries quickly and reliably.

We demonstrated how this technique is flexible enough to handle traditional statistical measures, such as quartiles, as well as highly customized percentile requirements, including quintiles or single arbitrary percentiles like the 90th. The ability to switch effortlessly between these modes simply by adjusting the `probs` vector makes this approach a versatile tool in the **data science** toolkit. Analysts should always remember to clearly define the probabilities of interest and assign meaningful names to the resulting summary columns for clear communication of results.

In summary, whenever an analysis requires understanding the internal distribution of a numerical variable conditional on a categorical factor, the methodology presented here--loading **dplyr**, using the pipe operator, and chaining `group_by()` with `summarize(quantile())`--offers the most effective and idiomatic solution in the R ecosystem. This ensures that distributional insights are derived accurately and efficiently across all defined subgroups.