Calculating Relative Frequencies in R with dplyr: A Step-by-Step Tutorial

Authored by

Mohammed looti

November 7, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Calculating Relative Frequencies in R with dplyr: A Step-by-Step Tutorial*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=12442

Mastering Relative Frequencies in Data Analysis with R

In advanced R programming and statistical inquiry, a recurring need arises: calculating the relative frequencies, or proportions, of specific categorical values within a given dataset. Calculating the relative frequency provides fundamental insight into the underlying distribution of observations, clearly illustrating the percentage contribution of each category to the total sample size. This essential process of converting raw counts into standardized, comparable metrics is a non-negotiable step in effective **Exploratory Data Analysis (EDA)**.

Although base R offers traditional methods for deriving these statistics, leveraging the highly optimized functions within the dplyr package--a foundational element of the Tidyverse ecosystem--significantly simplifies and accelerates the workflow. The `dplyr` package is celebrated for its powerful, readable syntax, which utilizes the concept of piping (`%>%`) to enable analysts to execute intricate data manipulation tasks in a transparent, sequential manner. This comprehensive tutorial is specifically designed to demonstrate how to efficiently use **dplyr's key verbs**--namely `group_by()`, `summarise()`, and `mutate()`--to accurately determine relative frequencies across various scenarios.

Understanding these proportional distributions is critical for a wide array of applications, spanning from targeted market segmentation strategies to rigorous quality control auditing. For instance, determining that 60% of product returns are tied to a single manufacturing defect provides management with a clear, prioritized area for immediate improvement. Calculating **relative frequencies** serves as the foundational layer for subsequent visualizations and complex modeling, as it ensures that all measures are standardized, independent of the overall sample size. We will commence by preparing our working environment and defining a representative sample dataset to be utilized throughout the following code examples.

Setting Up the R Environment and Defining the Sample Data Frame

Before we can dive into the calculations, it is imperative to ensure that the required libraries are loaded and our sample data is structured correctly. While the full Tidyverse is often loaded, we specifically rely on the functionality of the dplyr library. We will define a sample data frame that will act as the source for our analysis. This illustrative dataset is constructed to mirror common data structures encountered in organizational or sports statistics, featuring clear categorical variables such as `team` and `position`, alongside a quantitative measure like `points`.

The deliberate process of creating and reviewing this data frame ensures that our code examples are fully reproducible and clear for readers to follow. Our dataset is small, containing seven observations, which conveniently allows for easy manual verification of the calculated proportions. The precise structure and organization of the data frame are vital, as subsequent **dplyr**

operations depend heavily on correctly identifying the columns designated for grouping and aggregation.

The following code block outlines the steps used to generate our sample data frame, named `df`, along with the resulting output structure as seen in the R console. Observe the distinct distribution of observations between **Team A** and **Team B**; this distribution will be the initial focus of our frequency calculations.

#create data frame
df <- data.frame(team = c('A', 'A', 'A', 'B', 'B', 'B', 'B'),
position = c('G', 'F', 'F', 'G', 'G', 'G', 'F'),
points = c(12, 15, 19, 22, 32, 34, 39))

#view data frame
df

team position points
1 A G 12
2 A F 15
3 A F 19
4 B G 22
5 B G 32
6 B G 34
7 B F 39

Calculating Overall Relative Frequencies for a Single Variable

Our first analytical goal is to ascertain the overall **relative frequency** for each level within a single categorical column--specifically, the `team` variable. This calculation addresses the question: "What percentage of all total observations belong to Team A versus Team B?" To successfully derive this information, we will utilize a sequential pipeline of **dplyr operations** designed to transform the raw data into a clear summary table of proportions.

The operational sequence begins with the `group_by()` function, which prepares the data for aggregation based on the specified column (`team`). Immediately following the grouping, the `summarise()` function executes the calculation of the row count (`n`) for each defined group using the built-in R function `n()`. At this intermediate stage, the resulting table contains the **absolute frequencies** (raw counts) for each team. The crucial final step employs the `mutate()` function, which is used to generate a new column, `freq`. This new variable is calculated by dividing the count (`n`) of the current group by the total number of observations (`sum(n)`). It is noteworthy that

because `mutate()` follows `summarise()` in this pipeline, the `sum(n)` calculation automatically refers to the total count across all groups combined, thereby yielding the overall proportion.

The resulting summary table explicitly details the proportional breakdown: Team A, with 3 observations, constitutes approximately **42.9%** of the total dataset, while Team B, with 4 observations, accounts for the remaining **57.1%**. These proportional measures are profoundly important because they provide a standardized measure of distribution, which makes comparisons across varying datasets or time series much simpler. We can confidently conclude that Team B contributes slightly more than half of the total observations in our sample data frame.

The following code block captures this powerful, three-step data manipulation sequence, resulting in a clean and informative summary table:

library(dplyr)

df %>%
group_by(team) %>%
summarise(n = n()) %>%
mutate(freq = n / sum(n))

A tibble: 2 x 3
team n freq

1 A 3 0.429
2 B 4 0.571

The output confirms that Team A accounts for 42.9% of all rows in the data frame, and Team B accounts for 57.1% of rows. Critically, these proportions sum precisely to **1.0** (or 100%), which serves as a necessary verification step confirming that the calculation of relative frequencies across all categories is both exhaustive and mathematically correct. Understanding this fundamental operation is the gateway to tackling more sophisticated conditional frequency calculations.

Related Topic: For a deeper dive into aggregating data, explore techniques on The Complete Guide: How to Group & Summarize Data in R.

Deriving Conditional Proportions Across Multiple Variables

In many real-world analytical scenarios, simple overall proportions are inadequate. Data professionals frequently require **conditional proportions**, which represent the relative frequency of one variable calculated strictly *within* the context of another variable. For our sample dataset,

this involves calculating the proportion of player positions (`position`) relative to the total size of their respective teams (`team`). This refined calculation answers questions like: "Among all players on Team A, what percentage hold position F versus position G?"

The methodology used here is structurally identical to the previous example, but we expand the `group_by()` step to include both categorical variables: `team` and `position`. By grouping by both columns, we establish distinct, granular sub-groups (e.g., Team A, Position F; Team B, Position G; etc.). The subsequent `summarise(n = n())` step then calculates the raw count for every unique combination of team and position.

The most important distinction occurs within the `mutate(freq = n / sum(n))` operation. Since the data remains logically grouped by `team` at this stage (a specific behavior of `summarise()` when followed by other verbs), the `sum(n)` function calculates the total count **only within the currently processing team group**, not the grand total of the entire data frame. This preserved grouping context is precisely what allows us to derive true conditional probabilities--the frequency of a position relative to its team's specific total size.

The resulting data provides highly granular insights into the internal composition of each team. For instance, for Team A (which has a total N of 3), 2 players are in position F, yielding a 66.7% proportion within that team. Conversely, for Team B (total N of 4), 3 players are in position G, resulting in a 75.0% proportion within that team. These **conditional proportions** provide a substantially richer context than simple overall frequencies alone could offer.

Here is the implementation of the conditional frequency calculation pipeline:

library(dplyr)

df %>%
group_by(team, position) %>%
summarise(n = n()) %>%
mutate(freq = n / sum(n))

A tibble: 4 x 4
Groups: team
team position n freq

1 A F 2 0.667
2 A G 1 0.333
3 B F 1 0.250
4 B G 3 0.750

From the resulting table, we can extract the following specific conclusions regarding the internal structure of the teams:

66.7% of players on team A are in position F.
33.3% of players on team A are in position G.
25.0% of players on team B are in position F.
75.0% of players on team B are in position G.

Crucially, notice that within each respective team (A or B), the calculated frequencies sum exactly to 1.0, thereby confirming they are mathematically sound conditional proportions based on their group totals.

Related Topic: For guidance on how to create and transform variables using `mutate()`, refer to How to Use Mutate to Create New Variables in R.

Practical Application: Formatting Results as Clear Percentages

While decimal proportions (e.g., 0.667) possess the highest level of mathematical precision, reporting results to stakeholders or non-technical audiences often necessitates converting these figures into easily digestible **percentages** (e.g., 67%). The combined power of **dplyr** and base R functions allows us to embed this necessary formatting step directly within the data pipeline, ensuring the final output is immediately presentation-ready.

The conversion process requires modifying the expression used within the `mutate()` function. Instead of the simple calculation `n / sum(n)`, we must first multiply the proportion by 100. Next, we use the `round()` function to specify the desired level of precision (here, rounding to 0 decimal places). Finally, we employ the `paste0()` function to concatenate the resulting number with the percentage symbol (`%`). This seamless sequence transforms the numeric frequency into a clean, descriptive **character string**.

This integrated formatting technique is invaluable for generating automated summary reports where clarity, accuracy, and readability are paramount goals. By embedding the formatting logic within the **data manipulation pipeline**, we eliminate the need for subsequent manual steps and guarantee that the percentage labels are correctly and automatically associated with their corresponding counts and groups. This integration perfectly showcases the efficiency and practical utility of the **Tidyverse approach** to data processing and reporting.

The following code demonstrates how to apply this layer of formatting to the conditional frequency calculation derived in the previous example:

library(dplyr)

```
df %>%
group_by(team, position) %>%
summarise(n = n()) %>%
mutate(freq = paste0(round(100 * n/sum(n), 0), '%'))

# A tibble: 4 x 4
# Groups: team
team position n freq

1 A F 2 67%
2 A G 1 33%
3 B F 1 25%
4 B G 3 75%
```

A crucial consequence of utilizing `paste0()` is that the resulting `freq` column is implicitly converted from a numeric type to a **character type**, as indicated by the resulting output table. While this is ideal for direct visualization or reporting, analysts must be aware that this character column cannot be used directly in subsequent mathematical calculations. If further quantitative analysis is required, the best practice is often to create two separate columns: one containing the raw numeric proportion (as in Example 2) and another containing the formatted character percentage (as demonstrated here).

The Statistical Significance of Normalization and Relative Frequencies

The skill to rapidly and accurately calculate relative frequencies stands as a core competency in effective data analysis. Relying solely on **absolute counts** (`n`) can be highly misleading when attempting to compare categories, particularly if the total underlying group sizes are unequal. Consider a scenario where Organization X has 10,000 employees and records 50 safety incidents, while Organization Y has only 100 employees and records 3 incidents. The absolute counts suggest Organization X has far more incidents. However, the relative frequencies (0.5% vs. 3.0%) clearly reveal that **Organization Y** faces a significantly higher proportional safety risk.

Relative frequencies resolve this issue by reliably stabilizing comparisons through the process of **normalizing the data** based on the relevant total or the conditional group size. This standardization procedure is essential for establishing objective baselines, identifying statistical outliers, and conducting rigorous statistical testing. Furthermore, these proportions form the primary numerical input required for generating common, impactful visualizations, such as pie charts, stacked bar plots, and mosaic plots, which are indispensable tools for communicating complex data patterns simply and effectively to any audience.

By mastering the efficient **dplyr pipeline** demonstrated throughout this tutorial--the powerful combination of `group_by()`, `summarise()`, and `mutate()`--data professionals can confidently move beyond simple counting and begin engaging directly with the crucial distributional characteristics of their data. This specific skill set is absolutely indispensable for anyone working extensively with **categorical data** and striving to extract meaningful, actionable, and normalized insights from raw observations.