How to perform stratified sampling in R?

Authored by

stats writer

December 23, 2025

Stratified sampling is an essential statistical technique used to ensure that a research sample accurately reflects the diversity present in the overall population. Unlike simple random sampling, which might unintentionally over- or under-represent certain subgroups, stratification guarantees balanced representation. In the context of data analysis using the R programming language, executing this method efficiently requires robust tools. While base R offers functionalities, modern data manipulation packages significantly simplify the process, making complex sampling procedures accessible and reliable.

The core concept involves dividing the target population into non-overlapping groups, known as **strata**. These strata must be homogeneous internally but heterogenous externally. For instance, if studying student performance, grade level (Freshman, Sophomore, etc.) or gender could serve as effective strata. Once these groups are defined, researchers perform independent random sampling within each stratum. The final sample is then the aggregation of these smaller, independent samples, resulting in a highly representative subset of the data.

Historically, achieving stratified sampling in base R often involved cumbersome indexing or using specialized functions like `sample()`, potentially requiring complex calculations to determine the proportional sizes. However, the modern dplyr package, part of the tidyverse ecosystem, offers intuitive functions such as `group_by()`, `sample_n()`, and `sample_frac()`, which streamline the selection process dramatically. This tutorial will focus on leveraging these powerful tools to perform two common forms of stratification: sampling by fixed count and sampling by proportional fraction.

The Importance and Methodology of Stratified Sampling

Researchers frequently employ stratified sampling when the underlying population exhibits inherent variability that must be accounted for in the study design. When we take a sample from a larger population, the goal is often to use the data derived from that subset to draw reliable, unbiased conclusions about the population as a whole. If key demographic or structural characteristics are ignored during sampling, the resulting inferences may suffer from significant sampling bias, potentially invalidating the study's findings.

The procedure begins by identifying the stratification variable--the characteristic used to define the subgroups. In R, this variable must typically be treated as a factor variable, which categorizes the data points into distinct levels. After defining the strata, the sample size within each stratum can be determined either proportionally (where the sample size mirrors the stratum's size relative to the total population) or equally (where the same fixed number of units is drawn from every stratum, often used when researchers want equal statistical power across all groups).

A crucial advantage of using stratified sampling over simple random sampling is the reduction in sampling error, especially when the strata are strongly correlated with the variable of interest. For example, if we expect educational attainment to significantly affect income, stratifying by education

level ensures that each level is adequately represented, leading to more precise estimates of population parameters. This systematic approach guarantees that the sample is fully representative of the population regarding the variables used for stratification, enhancing the validity and generalizability of the research findings.

While the underlying statistical principles remain constant, the implementation in R has evolved. Modern data science practices emphasize readability and efficiency, leading to the widespread adoption of the dplyr package for data wrangling tasks, including complex sampling logic. By combining the `group_by()` function (to define the strata) and specific sampling functions (`sample_n()` or `sample_frac()`), R users can execute sophisticated stratified sampling commands with minimal, clear syntax, simplifying complex statistical operations.

Setting Up the Environment and Example Data in R

To demonstrate stratified random sampling effectively, we will use a common scenario: sampling students from a high school population. Imagine a high school comprising 400 students distributed across four grade levels: **Freshman**, **Sophomore**, **Junior**, and **Senior**. Our objective is to generate a stratified sample of 40 students, ensuring that we select exactly 10 students from each specific grade level. This structured approach mirrors many real-world survey methodologies where balanced representation across groups is mandatory.

Before proceeding with the sampling mechanism, we must first simulate the population data frame in R. Using `set.seed(1)` is a best practice, as it ensures that the random sampling results are reproducible, allowing others to verify the code and results exactly. We will create a data frame, `df`, containing two key variables: `grade` (our stratification variable, which is a factor variable) and `gpa` (a simulated numerical variable representing the students' Grade Point Average).

The simulation assumes an equal distribution, meaning 100 students for each of the four grade levels (4 × 100 = 400 total students). The `gpa` variable is simulated using a normal distribution centered around a mean of 85 with a standard deviation of 3, providing realistic, continuous data for analysis later on. The following R code illustrates the setup process, which is foundational for all subsequent sampling steps and confirms the initial structure of our synthetic population dataset.

#make this example reproducible
set.seed(1)

#create data frame for a population of 400 students
df <- data.frame(grade = rep(c('Freshman', 'Sophomore', 'Junior', 'Senior'), each=100),
gpa = rnorm(400, mean=85, sd=3))

#view first six rows of data frame to confirm structure and data types

```
head(df)

  grade      gpa
1 Freshman 83.12064
2 Freshman 85.55093
3 Freshman 82.49311
4 Freshman 89.78584
5 Freshman 85.98852
6 Freshman 82.53859
```

Inspecting the output of `head(df)` confirms that our data frame has been correctly initialized, showing the categorical `grade` variable and the continuous `gpa` variable. With this simulated population data ready, we can now proceed to the practical application of stratified sampling using the functions provided by the `dplyr` package in R. This preparation step ensures that the foundation for our sampling procedure is sound and reproducible.

Stratified Sampling by Fixed Row Count using sample_n()

When the research design requires drawing a specific, predetermined number of observations from each stratum, the `sample_n()` function from the dplyr package is the appropriate tool. This method is often preferred when researchers need to ensure equal sample sizes across different categories, even if the underlying population sizes are unequal. In our high school example, we explicitly requested 10 students from each of the four grade levels, totaling a sample size of 40. This approach guarantees equal statistical power for subgroup comparisons.

The key to performing this operation in `dplyr` lies in the combination of the piping operator (`%>%`) with the `group_by()` function. First, `group_by(grade)` instructs R to treat the data frame not as a single unit, but as four independent subgroups defined by the levels of the `grade` factor variable. Any subsequent operation will then be applied separately and independently to each group, which is the definition of stratification.

Immediately following the grouping operation, the `sample_n(size=10)` function is applied. Because the data is grouped, this command does not select 10 rows from the entire 400-row data frame; instead, it selects 10 random rows independently from the Freshman group, 10 from the Sophomore group, 10 from the Junior group, and 10 from the Senior group. This elegant two-step process achieves the precise requirements of fixed-count stratified random sampling with minimal code complexity.

The resulting data frame, `strat_sample`, contains the final subset of the population. We can verify the success of the stratification by using the `table()` function, which counts the frequency of

observations within each grade level in the final sample. The output must confirm that exactly 10 students were selected from each group, demonstrating successful, balanced stratification essential for comparative analysis between the grade levels.

library(dplyr)

#obtain stratified sample by specifying a fixed count (n=10) for each group
strat_sample <- df %>%
group_by(grade) %>%
sample_n(size=10)

#find frequency of students from each grade to verify the stratification
table(strat_sample$grade)

Freshman Junior Senior Sophomore
10 10 10 10

Stratified Sampling by Proportion using sample_frac()

In many statistical studies, it is more desirable to select samples proportionally to the size of the strata within the overall population. This ensures that the demographic distribution of the sample closely mirrors that of the population, leading to unbiased estimates of population totals or means. When aiming for proportional allocation in R using `dplyr`, the `sample_frac()` function is used instead of `sample_n()`.

The procedure remains structurally similar to the fixed-count method: the data frame is piped into `group_by()` to delineate the strata, and then the sampling function is applied. However, instead of specifying an integer count (`size=10`), we specify a fraction or percentage (e.g., `size=0.15` for 15%). This fraction represents the desired ratio of the stratum size that should be included in the final sample, maintaining proportionality.

For our example, suppose we decide that we want our sample to include 15% of the students from each grade level. Since our population is perfectly balanced (100 students per grade), selecting 15% from each group means we will draw 15 students (15% of 100) from the Freshman, Sophomore, Junior, and Senior classes, resulting in a total sample size of 60 students. This guarantees that the internal proportionality of the sample reflects the population structure exactly.

Using `sample_frac()` is particularly useful when the strata sizes are highly unequal. For instance, if 70% of the population belonged to Stratum A, proportional sampling would ensure that approximately 70% of the final sample also comes from Stratum A, maintaining representativeness that fixed-count sampling would violate. The code below demonstrates how to implement

proportional stratified random sampling in R.

```
library(dplyr)

#obtain stratified sample by specifying a fraction (15%) of rows for each group
strat_sample <- df %>%
group_by(grade) %>%
sample_frac(size=.15)

#find frequency of students from each grade to verify the proportionality
table(strat_sample$grade)

Freshman Junior Senior Sophomore
15 15 15 15
```

Comparing Fixed Count vs. Proportional Stratification

Choosing between `sample_n()` (fixed count) and `sample_frac()` (proportional fraction) depends entirely on the research objectives and the characteristics of the population. Fixed count sampling is ideal when the goal is to conduct detailed subgroup analysis or comparisons, requiring sufficient statistical power within smaller groups. For instance, if Freshman students represented only 5% of the total population, simple proportional sampling might only yield a handful of Freshman students, making statistical analysis for that group unreliable. Using fixed count ensures robust data for all strata.

Conversely, proportional stratification is essential when the primary objective is to make accurate estimations about the entire population. By maintaining the population's proportions, the resulting sample data can often be analyzed without complex weighting adjustments, yielding estimates that are highly representative of the overall demographic structure. This method minimizes sampling variance when the stratification variable is strongly related to the outcome variable, maximizing efficiency.

Regardless of the chosen method, both approaches rely on the powerful data manipulation capabilities of dplyr in R. The use of the `group_by()` command transforms the data processing flow, enabling researchers to apply random sampling processes simultaneously and independently across defined subgroups, thus automating what would otherwise be a tedious and error-prone manual operation of subsetting and sampling.

Advanced Considerations: Stratifying by Multiple Variables

While our examples used a single stratification variable (`grade`), stratified sampling can be

performed across multiple variables simultaneously. For instance, a researcher might wish to stratify not just by grade level, but also by gender (e.g., Freshman Males, Freshman Females, Sophomore Males, etc.). This refinement creates even finer subgroups, further reducing potential sampling bias, although it requires a larger overall sample size to ensure adequate representation in every resulting cell.

In dplyr, stratifying by multiple variables is straightforward: simply include all relevant factor variable names within the `group_by()` function. For example, `df %>% group_by(grade, gender) %>% sample_n(size=5)` would sample 5 students from every unique combination of grade and gender present in the data. Researchers must exercise caution, however, as too many stratification variables can lead to strata with very few or zero members, a situation known as 'empty cells,' which undermines the stratification process.

The flexibility provided by the R ecosystem, particularly the integration of statistical concepts with seamless data wrangling tools like `dplyr`, makes it an ideal environment for implementing complex survey methodologies. Mastery of these functions allows analysts to move beyond simple random selection and deploy sophisticated, scientifically sound sampling strategies tailored to specific research questions, leading to more robust and reliable scientific outcomes.

Conclusion: Ensuring Representative Data Samples

The ability to perform clean, verifiable stratified random sampling is a cornerstone of rigorous statistical analysis. By dividing the population into meaningful strata and then sampling independently from those groups, we minimize sampling variance and maximize the representativeness of our final dataset relative to the underlying population structure.

This tutorial demonstrated that using the `group_by()` function in conjunction with either `sample_n()` for fixed counts or `sample_frac()` for proportional allocation provides an efficient and highly readable method for implementing this technique within the R programming language. These methods ensure that whether the goal is equal representation across subgroups or accurate proportional representation of the whole, the resulting sample is statistically valid and ready for robust inference.

For those seeking to expand their knowledge of statistical sampling beyond stratification, further exploration into related methods such as cluster sampling and systematic sampling, along with their practical implementation in R, is highly recommended. These techniques form the foundation for reliable data-driven decision-making in research and industry, ensuring that conclusions drawn from samples are accurate representations of the entire population.

Types of Sampling Methods
Cluster Sampling in R

Systematic Sampling in R