How to Easily Extract the First N Rows from a Data Frame in R

Authored by

stats writer

November 28, 2025

In the world of data analysis, particularly within the R programming language, efficient inspection of large datasets is a fundamental skill. When dealing with a data frame—R's primary structure for storing tabular data—it is often necessary to quickly examine the initial observations without loading the entire dataset into view. This process of selecting the first N rows is essential for data validation, structure verification, and routine sanity checks.

Fortunately, R provides multiple robust and idiomatic methods for this task. The selection of the appropriate method often depends on the user's preference and whether they are primarily working within Base R environments or leveraging the tools provided by the Tidyverse, such as the powerful dplyr package. Understanding the nuances of each technique allows data practitioners to write clearer, more efficient, and highly readable code for subsetting operations.

Overview of Row Selection Techniques

To effectively select the first N rows of a data frame in R, practitioners typically rely on one of the following three core strategies. These methods offer different levels of flexibility and integration with other R functionalities, ensuring that analysts can choose the tool best suited for their current workflow:

Method 1: Utilize the head() function from Base R. This is the simplest and most conventional approach, designed specifically for viewing the top section of any R object with minimal syntax overhead.
Method 2: Employ Standard Indexing from Base R. This method uses positional subsetting, offering granular control over both row and column selection simultaneously, making it ideal for highly specific extractions.
Method 3: Adopt the slice() function from the dplyr Package. A modern solution preferred in the Tidyverse framework, often used in conjunction with the pipe operator (%>%) for sequential and highly readable data manipulation pipelines.

The following blocks illustrate the fundamental syntax for each approach before diving into detailed examples.

Method 1: Use head() from Base R

head(df, 3)

Method 2: Use Indexing from Base R

df

Method 3: Use slice() from dplyr

library(dplyr)

df %>% slice(1:3)

Establishing the Sample Data Frame

For demonstration purposes, we will utilize a standard R data frame named `df`, representing hypothetical sports team statistics including team names, points scored, and assists made. This setup allows us to clearly observe the output generated by each function and compare their syntactic differences while performing the core task of selecting the first N rows.

 # Create the sample data frame for demonstration
df <- data.frame(team=c('A', 'B', 'C', 'D', 'E', 'F', 'G'),
points=c(99, 90, 86, 88, 95, 99, 91),
assists=c(33, 28, 31, 39, 34, 35, 40))

View the data frame structure
df

team points assists
1 A 99 33
2 B 90 28
3 C 86 31
4 D 88 39
5 E 95 34
6 F 99 35
7 G 91 40

Method 1: Utilizing the head() Function (Base R)

The `head()` function is arguably the most recognizable and straightforward method for inspecting the initial portion of an R object. It is a utility function included in the default installation of Base R and requires no additional package loading. Its primary purpose, derived from the Unix utility of the same name, is providing a quick glance at the top of a data structure to confirm data integrity, variable types, and initial values.

To use `head()`, the user must specify two main components: the data object (in our case, `df`) and, optionally, the number of rows (`n`) they wish to retrieve. The syntax is concise, focusing solely on

the object and the positional count. This simplicity ensures maximum efficiency when the goal is purely observation of the dataset's beginning.

A significant feature of `head()` is its built-in default behavior. If the numerical argument `n` is omitted entirely, the function automatically displays the first six rows of the data frame. This standard default is often sufficient for initial assessment and is consistent across various R data structures, making the function predictable and highly usable across different coding environments.

We now apply `head()` to our sample data frame, demonstrating both explicit selection of N rows and the use of the function's default setting.

Example 1: Demonstrating head(df, N)

When we need a specific number of rows, we pass that number as the second argument. Here, we specify `3` to retrieve the top three observations from our sample data frame, `df`. This returns a new data frame containing only the specified rows.

 # Select the first 3 rows of the data frame explicitly
head(df, 3)

team points assists
1 A 99 33
2 B 90 28
3 C 86 31

Conversely, if we utilize the **head()** function without providing the numerical argument, R defaults to selecting the first six rows of the data frame. This is a crucial shortcut for routine checks on moderately sized datasets, allowing for rapid visual confirmation of structure and content without explicit argument passing.

 # Select the first 6 rows of the data frame (using default behavior)
head(df)

team points assists
1 A 99 33
2 B 90 28
3 C 86 31
4 D 88 39
5 E 95 34
6 F 99 35

Method 2: Leveraging Base R Indexing

Positional indexing, often referred to as subsetting, is a cornerstone skill in R data manipulation. This technique uses square brackets () to extract elements based on their position within a data structure. For a data frame, the indexing syntax is structured around two dimensions: `df`. This structure demands explicit understanding of R's matrix-like data representation.

To select the first N rows, we utilize the numeric sequence operator (`:`) in the row position (the first argument before the comma). For example, `1:N` generates a sequence from the first observation up to the Nth observation. The key operational element here is the comma; leaving the second argument (the column position) empty signals to R that we intend to include all available variables in the subsetted result. Hence, `df` extracts rows 1, 2, and 3, along with every column.

The primary benefit of using indexing over functional approaches like `head()` is its inherent versatility. It allows users to perform highly customized subsetting, such as selecting only the first N rows for specific columns or even omitting the first M rows if needed. While the syntax is essential for complex Base R manipulations, it can occasionally be less expressive than Tidyverse verbs, requiring careful attention to ensure correct row and column alignment.

Example 2: Using Indexing for Precise Subsetting

We apply the standard indexing syntax to retrieve the first three rows of our sample data frame. By leaving the column argument empty, we ensure that the entire width of the data frame is preserved in the output.

 # Select the first 3 rows of the data frame, keeping all columns
df

team points assists
1 A 99 33
2 B 90 28
3 C 86 31

The true advantage of indexing is demonstrated when we require a subset that includes specific rows and columns simultaneously. By specifying a vector of desired column names in the second index position, we can filter the result vertically as well as horizontally. The following code snippet demonstrates how to retrieve the first three observations while limiting the output exclusively to the 'team' and 'points' variables:

 # Select first 3 rows of 'team' and 'points' columns only
df

team points
1 A 99
2 B 90
3 C 86

Method 3: Utilizing slice() from dplyr

For R users integrated into the modern Tidyverse ecosystem, the dplyr package offers `slice()`, a dedicated function for selecting rows based on position. The Tidyverse methodology prioritizes functional programming and chaining operations using the pipe operator (`%>%`), which `slice()` is optimally designed to support. While `head()` is primarily for viewing, `slice()` is a manipulation verb, suitable for integrating into a series of data transformations.

The implementation of `slice()` is highly intuitive: the data frame is piped into the function, and the desired row index range (e.g., `1:N`) is passed as the argument. This design ensures that the focus remains on the action being performed rather than on positional arguments, which is a hallmark of the `dplyr` package's readability goals. This approach often results in code that is easier to debug and understand for teams collaborating on data projects.

Furthermore, `dplyr` extends this functionality with helper functions like `slice_head()`. While `slice(1:3)` achieves the same result as `head(df, 3)` in a simple context, `slice_head()` allows users to specify the number of rows to return based on groups previously defined by the `group_by()` function. This makes `slice()` and its variants incredibly powerful when conducting exploratory data analysis on grouped data, far surpassing the capabilities of Base R's simple `head()` function.

Example 3: Using slice() within the Tidyverse Workflow

To utilize the **slice()** function, it is mandatory to first load the dplyr package using the `library()` command. Once available, the data frame `df` is passed using the pipe operator, and the desired range `1:3` is specified to extract the initial rows.

library(dplyr)

Select the first 3 rows of the data frame using the pipe operator
df %>% slice(1:3)

team points assists
1 A 99 33
2 B 90 28

3 C 86 31

Choosing the Right Method for Data Inspection

When selecting the method for retrieving the first N rows of a dataset, R users must weigh trade-offs between speed, flexibility, and integration into existing workflows. There is no single "best" method; rather, the most effective choice depends on the immediate requirements:

head() Function: This method is unparalleled in its simplicity and speed for quick, standalone inspection. It is available by default in Base R and should be the default choice when the user only needs a fast preview of the data structure and content.
Base R Indexing: While slightly more verbose than functional alternatives, indexing provides the ultimate control over both row and column subsetting. It is essential when the goal is not just to view the beginning of the data, but to extract a precise subset of both dimensions for further calculation or manipulation.
slice() Function (dplyr): This method is highly recommended for users who regularly build complex data manipulation pipelines. It seamlessly integrates with the piping syntax, dramatically improving code clarity and consistency when row selection is just one step in a longer sequence of operations (e.g., filter, group, then slice).

Mastering all three techniques ensures fluency in diverse R programming environments, allowing the analyst to apply the most appropriate and efficient tool for any data inspection task.