Learning to Display All Rows of an R Tibble: A Comprehensive Guide

Authored by

Mohammed looti

November 7, 2025

The efficient management and clear visualization of tabular data form the bedrock of modern data analysis in R. While the traditional data frame has historically served as the foundational structure for storing datasets, the introduction of the **tibble**, championed by the tidyverse collection of packages, marked a significant evolutionary step. A **tibble** is essentially a modernized and highly refined version of a data frame in R that implements a significantly smarter and more controlled print method. This refinement is absolutely critical for working efficiently with the immense datasets common in modern analytics, as it prevents the console from being overwhelmed by attempting to display every single row and column--a frustrating and often system-crippling default behavior of classic data frames.

The primary distinction between a base data frame and a **tibble** lies in how the object handles truncation and display. By design, when a **tibble** object is called in the console, it is configured only to show the first 10 rows and those columns that fit comfortably within the current console width. This behavior is intentional, reflecting a design philosophy focused on performance and clarity. It provides the analyst with an immediate, informative snapshot of the data's structure, type, and initial values without sacrificing system performance or demanding excessive screen space. This controlled output is a huge advantage when handling datasets that contain thousands or even millions of observations, ensuring that the user always retains control over their output environment and workflow.

The Default Behavior: A Controlled and Informative Display

The moment a **tibble** is created or loaded into the R session, its sophisticated print method automatically activates. This method is engineered to provide immediate, actionable feedback about the dataset's dimensions and variable types without generating unnecessary clutter. Unlike base data frames, which can lead to lengthy, unwieldy console output that is difficult to navigate, the **tibble** prioritizes brevity and clarity. It explicitly states the dimensions (rows x columns) at the top and clearly lists the data type of each column immediately beneath the column name, offering a quick diagnostic summary that is unavailable in standard data frames.

The built-in 10-row limit acts as a powerful safeguard against accidental console overflow. Consider the scenario of loading a large dataset containing 100,000 observations; if R were to attempt to print all of them, the console could freeze, become unresponsive, or require extensive scrolling, severely disrupting the analytical flow. The **tibble** addresses this technical challenge directly by showing only the initial rows. This allows the user to quickly confirm essential details: that the data loaded correctly, that the variables are structured as expected (e.g., verifying that a numeric column contains decimal values or that a character column holds strings), and that the first few entries look sensible before proceeding with resource-intensive transformations.

To effectively illustrate this standard behavior, we will construct a practical example. We will create

a moderately sized **tibble** comprising 80 rows and 2 columns, populated with random normal data. This size is intentionally chosen: it is small enough that displaying all rows would not crash the system, but large enough to reliably trigger the **tibble's** default truncation feature, thereby highlighting its utility in managing output volume. For efficient data creation and manipulation, we rely on the functions provided by the **dplyr** package, a foundational component of the tidyverse.

Setting Up the Environment and Example Data

To ensure that this example is entirely reproducible by any user, we must first load the necessary library and set a seed for the random number generator. The use of the command `set.seed(1)` guarantees that the sequence of random values generated by `rnorm(80)` will be identical every time the code is executed, a crucial practice in high-quality data science work that promotes transparency and verification. We then proceed to construct the **tibble**, which we name `data`, containing 80 observations across two variables, `a` and `b`.

When we inspect the resulting `data` object by simply typing its name into the R console, the default print behavior is immediately activated. As anticipated, only the first 10 rows are displayed, accompanied by a crucial line indicating precisely how many rows have been suppressed. This mechanism is the core feature that prevents console overflow when working with production datasets that may contain hundreds of thousands or even millions of records. The concise display allows for immediate verification of the object's creation without the performance penalty of rendering the entire dataset.

The following is the R code used for setup, followed by the resulting default output demonstrating truncation:

```
#load dplyr
library(dplyr)

#make this example reproducible
set.seed(1)

#create tibble
data <- tibble(a = rnorm(80),
b = rnorm(80))

#view tibble
data
```

```
# A tibble: 80 x 2
a b
```

1 -0.626 -0.569
2 0.184 -0.135
3 -0.836 1.18
4 1.60 -1.52
5 0.330 0.594
6 -0.820 0.333
7 0.487 1.06
8 0.738 -0.304
9 0.576 0.370
10 -0.305 0.267
... with 70 more rows

As clearly illustrated in the output block, despite the fact that our object contains 80 total observations, the default display limits us to the first 10 rows. Crucially, the message `# ... with 70 more rows` serves as a transparent indicator that the data is merely truncated for display purposes, not lost or filtered. This inherent feature is extremely valuable for quick inspections, enabling the analyst to verify the initial structure and content without being forced to wait for hundreds or thousands of lines of output to scroll past, maintaining high productivity.

Adjusting the View: Printing a Specific Subset

While the standard 10-row limit is generally optimal for performance and quick checks, there are specific analytical scenarios--such as detailed debugging, manual data validation, or specific reporting tasks--where viewing more than the default subset is absolutely necessary. Fortunately, the **tibble** print method is highly configurable, allowing for precise control over the output volume. We can explicitly instruct R to display a specific, custom number of rows using the built-in `print()` function and specifying the `n` argument. This technique grants granular control over the output, allowing for fine-tuned adjustments based on the immediate analytical need without resorting to viewing the entire dataset.

To print, for example, the first 20 rows of our `data` **tibble**, we simply pass the integer 20 to the `n` argument within the `print()` function call: `print(data, n=20)`. This command overrides the default setting, forcing the display of the desired number of observations while still maintaining the column truncation features if the console width is limited. This method is highly recommended when you need a snapshot slightly larger than 10 rows but still wish to avoid printing the entire dataset, which could be visually cumbersome even for moderately large tibbles.

A more idiomatic and often preferred approach within the tidyverse environment is to utilize the **pipe operator** (`%>%`). This operator chains operations together seamlessly, significantly enhancing code readability by allowing the analyst to pass the `data` object directly into the `print()` function

in a flowing sequence: `data %>% print(n=20)`. While both the direct `print()` call and the chained piping approach achieve the identical result, the piping method is typically favored for its clean, step-by-step logic, aligning better with the overall tidyverse workflow philosophy.

Below we demonstrate both the direct `print()` call and the chained piping approach to display 20 rows, showing the flexibility available to the analyst:

#print first 20 rows of tibble using direct print() call
print(data, n=20)

 # A tibble: 80 x 2
a b

1 -0.626 -0.569
2 0.184 -0.135
3 -0.836 1.18
4 1.60 -1.52
5 0.330 0.594
6 -0.820 0.333
7 0.487 1.06
8 0.738 -0.304
9 0.576 0.370
10 -0.305 0.267
11 1.51 -0.543
12 0.390 1.21
13 -0.621 1.16
14 -2.21 0.700
15 1.12 1.59
16 -0.0449 0.558
17 -0.0162 -1.28
18 0.944 -0.573
19 0.821 -1.22
20 0.594 -0.473
... with 60 more rows

#print first 20 rows of tibble using the pipe operator
data %>% print(n=20)

 # A tibble: 80 x 2

a b

1 -0.626 -0.569
2 0.184 -0.135
3 -0.836 1.18
4 1.60 -1.52
5 0.330 0.594
6 -0.820 0.333
7 0.487 1.06
8 0.738 -0.304
9 0.576 0.370
10 -0.305 0.267
11 1.51 -0.543
12 0.390 1.21
13 -0.621 1.16
14 -2.21 0.700
15 1.12 1.59
16 -0.0449 0.558
17 -0.0162 -1.28
18 0.944 -0.573
19 0.821 -1.22
20 0.594 -0.473
... with 60 more rows

The Definitive Override: Displaying All Rows with Inf

To definitively answer the question of how to print all rows of a **tibble**, we must leverage the concept of infinity within the `print()` function arguments. The `n` argument, which explicitly controls the maximum number of rows displayed, accepts the special value `Inf` (representing infinity) to signal to the print method that all observations should be printed, regardless of the **tibble's** actual size. This method provides an exhaustive, full view of the dataset, which is sometimes necessary for final data checks, audit trails, or preparation for export.

By setting `n = Inf`, we are explicitly instructing R to ignore the size limit imposed by the **tibble's** default print method. It is absolutely crucial, however, to use this command with extreme caution and judgment. While highly effective for small to medium datasets (such as our 80-row example), applying `n = Inf` to a dataset containing hundreds of thousands or millions of rows will instantly result in a massive and potentially crippling output that defeats the primary efficiency purpose of using a **tibble** in the first place. This technique should therefore be reserved strictly for validation

purposes on curated subsets or reasonably small production data.

As before, we utilize the clean syntax of the pipe operator, ensuring the `data` object is piped directly into the print command with the infinity parameter applied. This maintains the clean, functional flow characteristic of the **dplyr** workflow:

#print all rows of tibble
data %>% print(n=Inf)

 # A tibble: 80 x 2
a b

1 -0.626 -0.569
2 0.184 -0.135
3 -0.836 1.18
4 1.60 -1.52
5 0.330 0.594
6 -0.820 0.333
7 0.487 1.06
8 0.738 -0.304
9 0.576 0.370
10 -0.305 0.267
11 1.51 -0.543
12 0.390 1.21
13 -0.621 1.16
14 -2.21 0.700
15 1.12 1.59
16 -0.0449 0.558
17 -0.0162 -1.28
18 0.944 -0.573
19 0.821 -1.22
20 0.594 -0.473
21 0.919 -0.620
22 0.782 0.0421
23 0.0746 -0.911
24 -1.99 0.158
25 0.620 -0.655
26 -0.0561 1.77
27 -0.156 0.717
28 -1.47 0.910

29 -0.478 0.384
30 0.418 1.68
31 1.36 -0.636
32 -0.103 -0.462
33 0.388 1.43
34 -0.0538 -0.651
35 -1.38 -0.207
36 -0.415 -0.393
37 -0.394 -0.320
38 -0.0593 -0.279
39 1.10 0.494
40 0.763 -0.177
41 -0.165 -0.506
42 -0.253 1.34
43 0.697 -0.215
44 0.557 -0.180
45 -0.689 -0.100
46 -0.707 0.713
47 0.365 -0.0736
48 0.769 -0.0376
49 -0.112 -0.682
50 0.881 -0.324
51 0.398 0.0602
52 -0.612 -0.589
53 0.341 0.531
54 -1.13 -1.52
55 1.43 0.307
56 1.98 -1.54
57 -0.367 -0.301
58 -1.04 -0.528
59 0.570 -0.652
60 -0.135 -0.0569
61 2.40 -1.91
62 -0.0392 1.18
63 0.690 -1.66
64 0.0280 -0.464
65 -0.743 -1.12
66 0.189 -0.751
67 -1.80 2.09

68 1.47 0.0174
69 0.153 -1.29
70 2.17 -1.64
71 0.476 0.450
72 -0.710 -0.0186
73 0.611 -0.318
74 -0.934 -0.929
75 -1.25 -1.49
76 0.291 -1.08
77 -0.443 1.00
78 0.00111 -0.621
79 0.0743 -1.38
80 -0.590 1.87

The resulting output confirms that all 80 rows are now displayed in the console, completely eliminating the trailing message about truncated observations. Note that even when printing all rows, the **tibble** maintains its structural headers (A tibble: 80 x 2) and variable type notations, offering superior clarity compared to a fully printed base data frame.

Configuration Options for Persistent Printing

If the requirement to view more than the default 10 rows is a frequent necessity during a specific analysis session, constantly appending `n=20` or `n=Inf` to the `print()` function can become repetitive and interrupt the analytical flow. For such cases, R provides a mechanism for global configuration of **tibble** printing behavior using the powerful `options()` function. This approach sets a persistent default for the current R session, overriding the standard 10-row limit for all subsequent tibble prints unless a specific `print(n=...)` command is used to override it momentarily.

To adjust the maximum number of rows displayed by default for any **tibble**, analysts should modify the global option `tibble.print_max`. For instance, executing `options(tibble.print_max = 50)` will ensure that all tibbles are printed up to a maximum of 50 rows automatically. If the dataset exceeds 50 rows, truncation will resume, but at the new, higher threshold. If you desire a persistent option to display all rows, you can set `tibble.print_max = Inf`; however, as previously noted, this should be done with extreme caution, particularly when dealing with environments where large data objects are frequently loaded.

Furthermore, **tibbles** also handle column width and truncation intelligently. If your goal is to ensure that every column is always displayed, regardless of how narrow your console window is, you can modify the `tibble.width` option. Setting `options(tibble.width = Inf)` will force R to attempt

to display all columns, though this often results in severe wrapping and poor formatting if you have a great number of variables. Alternatively, for visual inspection of very wide datasets (those with many columns), utilizing the `View()` function--which opens the data frame or **tibble** in a separate, spreadsheet-like viewer--is often the cleanest and most user-friendly solution, offering scrolling capabilities that overcome console limitations.

Best Practices for Data Inspection in R

The decision of whether to print all rows of a dataset should always be contextual, based on the size of the data and the purpose of the inspection. While we have provided the technical command to achieve full output using `n = Inf`, it is vital to constantly recall the core philosophy of the **tibble**: efficient handling of large data. Printing millions of rows offers no practical analytical benefit; it merely consumes valuable system resources, slows down the workflow, and renders the console output unmanageable.

Instead of relying on full printing, proficient analysts should leverage other functions within the **dplyr** package that are specifically designed for targeted and safe data inspection. These methods provide a much more structured approach to auditing large datasets:

`head(data, 50)`: Used to view a specific number of rows (e.g., 50) from the beginning of the dataset.

`tail(data, 50)`: Used to inspect the last 50 rows of the dataset, which is useful for checking data integrity after appending new observations.

`slice(data, 100:150)`: Used to inspect an arbitrary, non-contiguous range of rows deep within the dataset without needing to print everything before it.

`glimpse(data)`: Provides a transposed summary of the data, listing variable names, data types, and the first few values in a highly compact format, regardless of the number of rows or columns.

In summary, while the default truncation behavior of **tibbles** represents a significant and welcome improvement over base data frames, flexibility remains paramount. When a complete view of a small to medium dataset is genuinely required, the `print(n=Inf)` command provides an immediate and effective override. For those new to tibbles and the tidyverse workflow, a crucial resource for mastering these techniques is the official documentation, such as the comprehensive tibbles chapter in *R for Data Science*.

You can find more R tutorials covering advanced data manipulation and visualization techniques here.