Descriptive Statistics in R: A Practical Guide Using `stat.desc()`

Authored by

Mohammed looti

November 12, 2025

In the demanding field of data analysis, obtaining a rapid, comprehensive summary of your datasets is not merely helpful--it is essential. This foundational process, formally known as calculating descriptive statistics, provides fundamental quantitative insights into the data's central tendency, dispersion, and overall distribution shape. Before commencing any complex modeling or inferential tests, analysts must first understand these basic characteristics.

Although the base installation of R provides a rich selection of statistical functions--such as `mean()`, `sd()`, `min()`, and `max()`--the task of manually executing these commands for every variable within a large data frame and then compiling the results into a unified, presentable table is inefficient and prone to error. Data practitioners frequently encounter large datasets where automation is crucial for reproducibility and speed.

To address this need for streamlined reporting, the extended R package ecosystem offers specialized tools. One of the most robust and versatile solutions for generating detailed, matrix-formatted descriptive statistics in R is the powerful **stat.desc()** function. This function is bundled within the specialized **pastecs** package, dramatically simplifying the often repetitive process of generating multiple statistical metrics across numerous quantitative variables simultaneously.

Mastering the stat.desc() Function Syntax

The **stat.desc()** function is designed for high flexibility, allowing users to precisely control the type and depth of statistical output they receive through a set of optional arguments. By default, the function generates a comprehensive suite of standard statistics. However, its true power lies in the ability to selectively include advanced metrics related to dispersion and normality testing.

Understanding the structure of the function call is the first step toward leveraging its full potential. The fundamental syntax structure for invoking this powerful function is defined by four primary parameters, each governing a different aspect of the calculation:

stat.desc(x, basic=TRUE, desc=TRUE, norm=FALSE, p=0.95)

Each argument listed above plays a critical role in determining which statistics are calculated and how precision statistics, like confidence intervals, are defined:

x: This is the only required argument. It specifies the name of the input object, typically a data frame or a simple vector, for which you wish to calculate the descriptive statistics. The structure of the output will depend on the dimensions of this object.
basic: This argument defaults to **TRUE**. When set to **TRUE**, it instructs the function to return the core, foundational statistical measures. These include counts (nbr.val), minimum (min), maximum (max), range, and the total sum of values (sum). These metrics provide an immediate sense of the data's bounds and volume.

desc: Also defaulting to **TRUE**, this parameter generates the more detailed summary statistics that are typically used for in-depth reporting. These include measures of location (median, mean), precision (standard error of the mean or SE.mean, and confidence interval for the mean or CI.mean), and dispersion (variance or var, and standard deviation or std.dev).
norm: Unlike the others, this argument is set to **FALSE** by default. If you set **norm=TRUE**, the function will calculate statistics specifically related to assessing the assumption of a normal distribution. This includes measures of asymmetry (skewness), tail heavy-ness (kurtosis), and the results of formal normality tests such as the Shapiro-Wilk test. This is essential for preparatory work before running parametric inferential tests.
p: This parameter is used to define the significance level when calculating the confidence interval values (CI.mean). The default value of 0.95 corresponds to a 95% confidence level. If you require a tighter or looser interval, such as 99%, you would set `p=0.99`.

By selectively adjusting these parameters, analysts can generate highly customized statistical summaries tailored to their specific analytical needs. The following sections will provide a complete, hands-on example, demonstrating how to prepare data and effectively leverage the **stat.desc()** function within your R environment, addressing the crucial challenge of handling mixed data types.

Setting Up Your R Environment: Data Preparation

To demonstrate the practical utility and versatility of **stat.desc()**, we will first create a small, representative dataset. Imagine a scenario in sports analytics where we are tracking fundamental performance metrics for a group of basketball players. Our data frame will include a categorical identifier (team affiliation) and two numerical performance metrics (total points scored and total assists accumulated).

Before executing the statistical calculation, two steps are required: defining the sample data frame and ensuring that the necessary package, **pastecs**, is installed and loaded into the active R session, as the **stat.desc()** function is not part of the standard base R installation. We use R's data frame creation and viewing commands as follows:

#create data frame
df <- data.frame(team=c('A', 'A', 'A', 'A', 'B', 'B', 'B', 'B'),
points=c(22, 39, 24, 18, 15, 10, 28, 23),
assists=c(3, 8, 8, 6, 10, 14, 8, 17))

#view data frame
df

team points assists

1 A 22 3
2 A 39 8
3 A 24 8
4 A 18 6
5 B 15 10
6 B 10 14
7 B 28 8
8 B 23 17

With our sample data, named `df`, now properly structured, we can proceed directly to the core task: calculating the descriptive statistics for every column using the **stat.desc()** function. The next example demonstrates what happens when we pass the entire data object to the function without modification.

Analyzing Mixed Data Types: The Default Output

Our initial application of **stat.desc()** will involve passing the entire data frame, `df`, to the function using all default settings (`basic=TRUE` and `desc=TRUE`). This approach provides an excellent illustration of the function's behavior when it encounters a mix of quantitative (numeric) and qualitative (character or factor) variables within the same object.

library(pastecs)

#calculate table of descriptive statistics for each column in data frame
stat.desc(df)

team points assists
nbr.val NA 8.0000000 8.000000
nbr.null NA 0.0000000 0.000000
nbr.na NA 0.0000000 0.000000
min NA 10.0000000 3.000000
max NA 39.0000000 17.000000
range NA 29.0000000 14.000000
sum NA 179.0000000 74.000000
median NA 22.5000000 8.000000
mean NA 22.3750000 9.250000
SE.mean NA 3.0991790 1.566958
CI.mean NA 7.3283939 3.705267
var NA 76.8392857 19.642857
std.dev NA 8.7658021 4.432026

coef.var NA 0.3917677 0.479138

The resulting output is a matrix that clearly presents the descriptive statistics. Each row corresponds to a specific metric (e.g., mean, median, Standard Deviation), and each column corresponds to a variable (`team`, `points`, `assists`). This format facilitates direct comparison of the key metrics between variables.

A significant feature of this output, and a crucial lesson in data preparation, is the status of the `team` column. Because `team` contains character data--non-numeric identifiers (A and B)--it is mathematically impossible to calculate measures of central tendency, dispersion, or sum for it. Consequently, **stat.desc()** intelligently returns **NA** (Not Available) for all metric values associated with the `team` variable. This behavior underscores the necessity of managing data types carefully when using quantitative analysis functions in R.

Best Practices: Subsetting for Clean Numerical Analysis

In nearly all real-world data analysis scenarios, a data frame will contain a mixture of different variable types. While quantitative variables (like `points` and `assists`) are the target for statistical metrics, categorical or identifying variables (like `team`) must be excluded from these calculations. Failing to do so results in the cluttered, informative-but-messy output seen above, filled with **NA** values.

To ensure a clean, actionable output, the accepted best practice is to subset the data frame before passing it to **stat.desc()**. This involves instructing the function to process only the numerical columns. For our basketball data, we are interested solely in the metrics for `points` and `assists`. We can achieve this precise selection using R's standard indexing capabilities, targeting only the columns by their names:

library(pastecs)

#calculate table of descriptive statistics for points and assists columns
stat.desc(df)

points assists
nbr.val 8.0000000 8.000000
nbr.null 0.0000000 0.000000
nbr.na 0.0000000 0.000000
min 10.0000000 3.000000
max 39.0000000 17.000000
range 29.0000000 14.000000
sum 179.0000000 74.000000

median 22.5000000 8.000000
mean 22.3750000 9.250000
SE.mean 3.0991790 1.566958
CI.mean.0.95 7.3283939 3.705267
var 76.8392857 19.642857
std.dev 8.7658021 4.432026
coef.var 0.3917677 0.479138

This refined, clean output provides a comprehensive set of descriptive statistics focused exclusively on the quantitative variables. This clarity allows for immediate and effective interpretation, revealing that the `points` variable is significantly more spread out (range of 29.0) than the `assists` variable (range of 14.0). Analysts should always prioritize subsetting to maximize the interpretability of their summary tables.

Deciphering the Output: A Deep Dive into 14 Metrics

The default output of the **stat.desc()** function is highly informative, providing 14 distinct rows of statistics that span measures of count, location, dispersion, and precision. To fully utilize this powerful function, it is critical to understand the definition and interpretation of each metric presented in the resulting table matrix.

The statistics presented below cover the full range of basic and descriptive metrics provided when both `basic=TRUE` and `desc=TRUE` are employed:

nbr.val: Represents the total number of non-missing (non-NA) observations or values found within the column. This confirms the size of the dataset used for the calculation.
nbr.null: This is the count of zero (0) values present in the column. It is important for distinguishing between missing data (NA) and legitimate zero observations.
nbr.na: Provides a count of missing values (Not Available) in the column. High NA counts indicate the variable may require imputation or careful handling before analysis.
min: The absolute smallest numerical value observed in the dataset.
max: The absolute largest numerical value observed in the dataset.
range: Calculated as the difference between the maximum and minimum values (max - min). This provides the simplest measure of the overall spread or variability of the data.
sum: The total aggregate of all numerical values in the column, useful for budget tracking or cumulative metrics.
median: The middle value of the dataset when all observations are ordered. It serves as a measure of central tendency that is robust to outliers, as 50% of the data falls below this point.
mean: The arithmetic average of the values, calculated by dividing the sum by the number of values. It is the most commonly used measure of central tendency.

S.E. mean: The Standard Error of the Mean. This metric estimates the variability between the sample mean and the true population mean, serving as a measure of the precision of the sample mean estimate.
CI mean .95: The 95% confidence interval for the mean value (note: the label adapts based on the `p` parameter, e.g., CI.mean.0.99 if `p=0.99`). This indicates the range within which the true population mean is likely to fall, with a specified degree of confidence.
var: The variance of the values, which measures the average squared deviation of each value from the mean. It is fundamental to many advanced statistical calculations.
std.dev: The Standard Deviation of the values. Calculated as the square root of the variance, it provides a highly interpretable measure of dispersion expressed in the original units of the data.
coef.var: The Coefficient of Variation. Calculated as the ratio of the Standard Deviation to the mean, this unit-less metric is highly useful for comparing the relative variability across different datasets, even if they have vastly different scales.

The comprehensive nature of **stat.desc()** makes it an indispensable tool for initial data reporting and exploration. Remember that flexibility is key; you can adjust the precision of the output by altering the **p** argument, such as setting `p=0.99` to generate the 99% confidence interval, which would be displayed as CI.mean.0.99 in your final table.

Additional Resources

The following tutorials explain how to perform other common tasks in R:

Featured Posts

5 Statistical Biases to Avoid

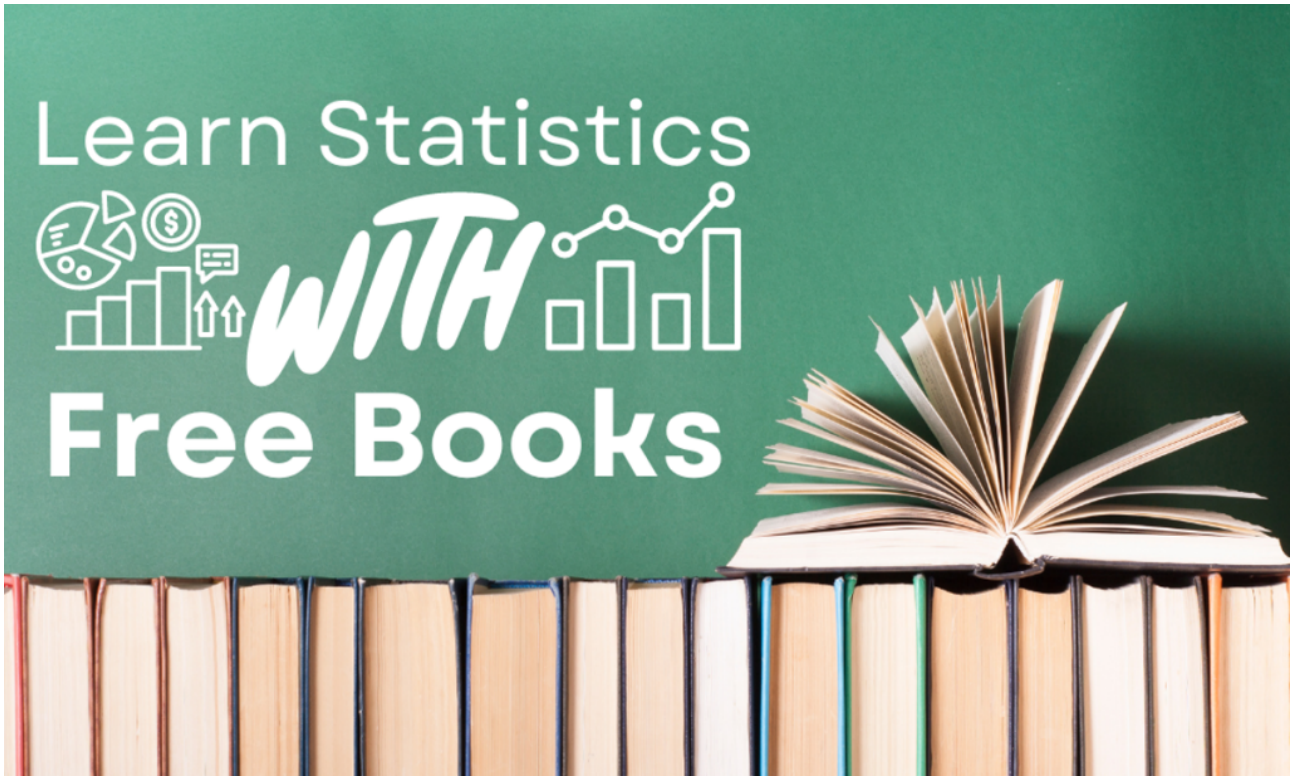
April 25, 2024

[5 Free Statistics Courses for Beginners]

April 19, 2024

5 MIT Statistics Courses That Are Free

April 18, 2024



[5 Free Books to Learn Statistics]

April 18, 2024

How to Use the info() Method in Pandas

April 12, 2024

How to Use pct_change() in Pandas

April 12, 2024