How to use the CMISS Function in SAS?

Authored by

stats writer

November 19, 2025

The **CMISS function** is a highly valuable utility within SAS, designed specifically for efficient identification and quantification of missing data across observations. Unlike calculating simple totals for an entire variable, CMISS function operates row-wise, providing an immediate count of how many arguments (variables) provided to it contain a missing value for that specific record. This capability is fundamental for robust data quality checks and subsequent imputation strategies in advanced data analysis projects.

Understanding the distribution of missingness is crucial; if certain rows consistently show a high count of missing values, it might indicate a systemic data collection error or an observation that should be excluded from specific analyses. The CMISS function simplifies this diagnostic process, returning a single numeric value representing the total count of missing fields for the variables specified in the function call for the current observation being processed within the DATA step. This function is instrumental when dealing with large volumes of data where manual inspection of every record is infeasible, ensuring high standards of data integrity are maintained throughout the analytical pipeline.

Fundamentals of the CMISS Function

The core purpose of the **CMISS** function is to count the number of missing values among a list of variables or expressions provided as arguments. When used within a DATA step, this function is typically employed to create a new variable that summarizes the integrity of each row in the input dataset. This process enables analysts to quickly triage data based on completeness, which is often a prerequisite for rigorous modeling and reliable statistical inference.

A common application involves specifying a range of variables using the powerful double-dash notation (`--`) or the `OF` keyword, which allows the function to efficiently evaluate many columns without listing each one individually. This structure is particularly useful when working with large datasets containing dozens or hundreds of variables, where manual enumeration would be cumbersome and prone to error. The resulting count is then stored in a newly generated variable, often named something descriptive like **total_missing** or **missing_count**, which can then be used for filtering or conditional processing.

Consider the general structure utilized for integrating **CMISS** into a transformation script. The following example illustrates the concise syntax that allows SAS to calculate the row-wise missing value count across a defined set of columns, focusing specifically on a range of variables within the current observation:

```
data new_data;
set my_data;
total_missing = cmiss(of team -- assists);
run;
```

In this specific syntax, a new dataset, named `new_data`, is generated based on the source `my_data`. Crucially, the new variable `total_missing` is populated by counting the missing observations found across the entire range of variables starting with `team` and ending with `assists`, based on their sequential order in the input dataset. This demonstrates the efficiency of using variable list shortcuts when defining the scope of the **CMISS** evaluation, significantly simplifying the code required for comprehensive checks.

Syntax Breakdown and Variable List Specification

The power of CMISS function often lies in its flexibility regarding argument specification. While you can list individual variables separated by commas, the preferred method for assessing data completeness across a wide range of variables involves leveraging SAS variable list conventions. These conventions significantly reduce coding complexity, improve script maintainability, and ensure that changes to the underlying dataset structure are handled efficiently.

There are several robust ways to define the list of variables for the **CMISS** function, allowing the analyst to tailor the scope of the missingness check precisely to the analytical requirements:

`CMISS(var1, var2, var3, ...)`: Explicitly listing each variable. This is suitable for a small, non-contiguous selection of columns where only specific metrics are critical to the analysis.

`CMISS(OF var_start -- var_end)`: Using the double-dash range notation. This includes all variables physically located between `var_start` and `var_end` in the input dataset structure. This is highly efficient but requires knowledge of the physical order of variables.

`CMISS(OF _NUMERIC_)` or `CMISS(OF _CHARACTER_)`: Using special keyword lists. These options allow the function to automatically evaluate all numeric variables or all character variables currently defined in the DATA step, respectively. Utilizing these system-defined lists is highly effective for comprehensive completeness checks across a specific data type.

It is important to remember how SAS handles missing values for different data types. Numeric missing values are represented by a period (`.`), while character missing values are represented by a blank (or an empty string). The **CMISS** function correctly identifies both types of missingness when calculating the row total, ensuring accuracy regardless of variable type within the specified range, which is a key advantage over functions that only operate on numeric arguments.

Illustrative Example: Basketball Player Data

To demonstrate the practical application of **CMISS**, we will use a hypothetical dataset containing performance metrics for various basketball players. This dataset, named `my_data`, intentionally includes several instances of missing data (represented by a period for numeric variables and

blanks/periods for the character variable in the input method shown), simulating common issues encountered in real-world data collection processes where certain observations may lack complete information.

We begin by generating the sample dataset using the DATA step and `DATALINES` statement. This establishes the foundation for our subsequent row-wise missing value analysis. Note the explicit representation of missing values (the period `.`) in the input data where critical metrics like `team`, `points`, or `assists` are unavailable. This intentional introduction of gaps allows us to verify the accuracy of the **CMISS** calculation.

Step 1: Creating the Sample Dataset

The following code block defines the `my_data` dataset, which includes three key variables: `team` (character), `points` (numeric), and `assists` (numeric). Careful inspection reveals that several observations suffer from one or more instances of missing information, which the **CMISS** function will be tasked with identifying and counting during the subsequent transformation phase.

```
/*create dataset*/
data my_data;
input team $ points assists;
datalines;
Cavs 12 5
Cavs 14 7
Warriors 15 9
. 18 9
Mavs 31 7
Mavs . 5
. . 3
Celtics 36 9
Celtics 40 7
;
run;

/*view dataset*/
proc print data=my_data;
```

Upon reviewing the output of the `PROC PRINT` statement, it becomes visually evident that this data contains rows with incomplete fields. For instance, the fourth row is missing the team name, while the sixth and seventh rows contain multiple missing values across both character and numeric fields. This initial review confirms the necessity of using a function like CMISS function for

automated data quality assessment, paving the way for targeted data cleaning.

The visualization below confirms the structure and the presence of missing data points represented by the default system missing indicator in the original dataset:

Obs	team	points	assists
1	Cavs	12	5
2	Cavs	14	7
3	Warriors	15	9
4		18	9
5	Mavs	31	7
6	Mavs	.	5
7		.	3
8	Celtics	36	9
9	Celtics	40	7

Step 2: Implementing CMISS to Count Missing Values

The next logical step is to introduce the **CMISS** function into the DATA step to generate our row-wise missing value count. We will create a new dataset, `new_data`, inheriting all original variables from `my_data`, and appending the crucial summary variable `total_missing`.

We employ the concise variable list notation `(of team -- assists)`. Since `team` is the first variable and `assists` is the last in our small dataset, this command effectively tells SAS to check all columns in the dataset for missingness. This ensures a complete assessment of the integrity of each record relative to the available performance metrics, regardless of whether the variables are numeric or character.

```
/*create new dataset that counts number of missing values in each row*/
data new_data;
set my_data;
total_missing = cmiss(of team -- assists);
run;
```

The execution of this DATA step results in the creation of `new_data`, where the added column serves as a powerful diagnostic tool. By examining the values in `total_missing`, we can immediately identify which observations require attention due to high levels of missing data, thus allowing for targeted follow-up actions like contacting data sources or applying sophisticated

imputation models.

Analysis of the Output Dataset

Once the new dataset is generated, viewing the output confirms the successful operation of the **CMISS** function. The new column, `total_missing`, quantifies the extent of data incompleteness on a per-row basis. This metric is instrumental in subsequent steps of data preparation, such as filtering incomplete observations or selecting appropriate imputation techniques based on the observed pattern of missingness.

The visualization below shows the `new_data` dataset, demonstrating how the calculated `total_missing` column integrates seamlessly with the original player data, providing a clear summary of record quality:

Obs	team	points	assists	total_missing
1	Cavs	12	5	0
2	Cavs	14	7	0
3	Warriors	15	9	0
4		18	9	1
5	Mavs	31	7	0
6	Mavs	.	5	1
7		.	3	2
8	Celtics	36	9	0
9	Celtics	40	7	0

Analyzing specific rows provides clarity on how **CMISS** correctly handles mixed missingness across different data types:

The first three rows (Cavs, Warriors) contain **0** missing values, indicating full completeness across the checked variables, resulting in perfect records.

The fourth row shows **1** missing value (the team name), confirming the function correctly identified the missing character variable.

The seventh row displays **2** missing values, corresponding to the missing team name and missing points value, highlighting severely incomplete records.

The remaining rows contain either **0** or **1** missing value, confirming the accuracy of the automated

count.

This detailed, row-by-row count facilitates immediate assessment of data quality, enabling analysts using SAS to make informed decisions about how to proceed with potentially flawed records, thereby enhancing the rigor of their data analysis.

Advanced Considerations and Resources

While **CMISS** is excellent for counting missing values, it is often paired with other SAS functions for advanced applications. For instance, you might use **CMISS** in conditional logic (`IF` statements) within a DATA step to exclude rows where `total_missing` exceeds a certain threshold (e.g., records missing more than 50% of their critical variables). Furthermore, remember that **CMISS** counts system-missing values (`.` for numeric, blank for character) but does not count special user-defined missing values (like `.A`, `.B`, etc.) unless the user defines logic to treat those special values as missing.

For tasks requiring comparison with other summary statistics functions, CMISS function is often contrasted with the **N function** (which counts non-missing numeric arguments) and the **NMISS function** (which counts missing numeric arguments only). It is critical to select **CMISS** when checking across mixed data types (character and numeric), as it provides the most comprehensive row-wise missingness check, making it the preferred tool for overall record integrity assessment.

For users seeking the complete technical specifications and additional optional arguments for the **CMISS** function, the official SAS documentation provides exhaustive details on its behavior, particularly concerning array handling and different types of missing values. Consulting these resources is essential for deploying the function effectively in complex data environments.

Note: You can find the complete documentation for the SAS **CMISS** function by visiting the official SAS website.

Related SAS Data Management Techniques

Efficient data management often requires familiarity with a suite of tools beyond just counting missing observations. Once missing values are identified using **CMISS**, the next steps typically involve strategies for handling them--such as imputation, deletion, or flagging. The choice of strategy heavily depends on the nature and extent of the missing data.

The techniques below often follow the application of **CMISS** in the initial stages of data preparation for sophisticated data analysis:

Imputation using PROC MEANS: Replacing missing values with the mean, median, or mode

calculated from the non-missing values of that variable. This is a simple technique often used when missing data is assumed to be Missing Completely at Random (MCAR).

Missing Value Flagging: Creating binary indicator variables (0/1) for each column where missingness was detected, allowing statistical models to account for the impact of the missing data itself, which is crucial if the data is Missing At Random (MAR).

Listwise Deletion: Removing entire observations where the **CMISS** count is greater than zero for key variables, though this is only recommended when the proportion of missing data is small to avoid significant loss of statistical power and potential bias.

These methods, coupled with the precision offered by the **CMISS function**, ensure that data cleaning is performed systematically and documented thoroughly before proceeding to advanced statistical modeling in SAS.

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