What is the MISSING Function in SAS (With Examples)

Authored by

stats writer

November 19, 2025

The MISSING function in SAS is an indispensable utility designed specifically to handle data quality issues by efficiently detecting and identifying missing values within a dataset. Dealing with missing data is a fundamental step in any data preparation workflow, as unprocessed nulls can skew statistical results and compromise model accuracy.

For instance, the MISSING function can be deployed to scrutinize a numeric variable, such as a customer's recorded age or purchase amount. Once detected, these missing numeric values (represented typically by a dot in SAS) can be imputed using appropriate measures like the mean, median, or zero. Furthermore, it is equally effective at identifying missing character values--often blank strings or single spaces--in fields like a customer's address or job title, allowing analysts to replace them with a standardized default value, such as 'unknown' or 'not specified'. Utilizing the MISSING function is a critical practice that ensures the resulting dataset is robust, complete, and fully prepared for rigorous statistical analysis.

Understanding the MISSING Function Syntax and Behavior

You can use the **MISSING** function within SAS programming statements, typically in a `DATA step`, to programmatically verify if a specified variable contains a missing value. This function is designed for simplicity and efficiency, returning a binary result based on its evaluation.

The general syntax required for calling this crucial data quality function is straightforward:

MISSING(expression)

where the components are defined as follows:

expression: This argument represents the specific variable (or any other valid SAS expression) whose value you wish to evaluate. The MISSING function is polymorphic, meaning it handles both character and numeric variables seamlessly.

When executed, this function returns an integer value indicating the presence or absence of a missing observation. Specifically, it returns **0** (representing false) if the evaluated expression or variable observation does not contain a missing value. Conversely, it returns **1** (representing true) if the observation being checked does contain a missing value. This binary output makes it exceptionally useful for direct use in conditional processing and filtering.

The following practical example demonstrates how to implement and leverage the output of this function within a typical SAS data processing workflow.

Implementing the MISSING Function in SAS: A Practical Example

To illustrate the power of the MISSING function, consider a hypothetical dataset we create in SAS. This dataset, named `my_data`, contains essential statistics and positional information for various basketball players, intentionally including several instances of missing values to simulate real-world data challenges.

```
/*create dataset containing player statistics*/
data my_data;
input team $ position $ points assists;
datalines;
A Guard 14 4
A Guard 22 6
A Guard 24 9
A Forward 13 8
A Forward 13 9
A . 10 5
B Guard 24 4
B Guard . 6
B Forward 34 2
B Forward 15 5
B Forward 23 5
B . 10 4
;
run;

/*view the initial dataset structure*/
proc print data=my_data;
```

Obs	team	position	points	assists
1	A	Guard	14	4
2	A	Guard	22	6
3	A	Guard	24	9
4	A	Forward	13	8
5	A	Forward	13	9
6	A		10	5
7	B	Guard	24	4
8	B	Guard	.	6
9	B	Forward	34	2
10	B	Forward	15	5
11	B	Forward	23	5
12	B		10	4

Detecting Missing Character Values

Our immediate goal is to systematically audit the `position` variable within this dataset. The `position` variable is a character type, and we observe that rows 6 and 12 contain missing values (represented by a single dot `.` in the input data, although SAS treats missing character values typically as blanks). We can create a new dataset, `new_data`, and utilize the **MISSING** function to generate an indicator variable that flags whether the `position` column is missing a value for each respective observation.

In the following `DATA step`, we create a new variable named `missing_position`. This variable is assigned the result of the `missing(position)` call. The resulting value will be 1 if the position is missing or 0 if it contains a recorded value.

```
/*create new dataset using the MISSING function*/
data new_data;
set my_data;
missing_position = missing(position);
run;

/*view the new dataset structure and results*/
proc print data=new_data;
```

Obs	team	position	points	assists	missing_position
1	A	Guard	14	4	0
2	A	Guard	22	6	0
3	A	Guard	24	9	0
4	A	Forward	13	8	0
5	A	Forward	13	9	0
6	A		10	5	1
7	B	Guard	24	4	0
8	B	Guard	.	6	0
9	B	Forward	34	2	0
10	B	Forward	15	5	0
11	B	Forward	23	5	0
12	B		10	4	1

Analysis of Binary Missing Value Indicators

Upon viewing the output of the `new_data` dataset, we can clearly observe the performance of the MISSING function. The newly generated column, labeled `missing_position`, operates as a binary flag: it holds a value of **0** when there is a valid, non-missing entry in the `position` column, and it displays a value of **1** precisely where a missing value was detected in the corresponding row of the source column.

A crucial observation: Note that row 8 in the original data contains a missing value specifically in the `points` column (a numeric variable). However, because we only applied the **MISSING** function to the `position` variable, the `missing_position` column correctly returns a value of **0** for this row, indicating that the position data itself is present. This highlights that the function operates exclusively on the single expression or variable provided as its argument, ignoring missingness in other fields.

Combining MISSING with Conditional Logic (IF-ELSE Statements)

While the binary output (0 or 1) provided by the **MISSING** function is highly effective for statistical calculations and filtering, analysts often prefer more descriptive, human-readable indicators, particularly when generating audit reports or preparing data for non-technical consumption. SAS facilitates this by allowing the integration of the **MISSING** function within IF-ELSE conditional statements. This powerful combination allows us to return custom character values, such as 'yes' or 'no', instead of the default numeric indicators.

The following code demonstrates how to use the MISSING function as the condition within an IF-ELSE block. Since the function inherently returns 1 (true) if the value is missing, the `THEN` clause executes automatically when missingness is detected.

```
/*create new dataset using IF-ELSE with MISSING function*/
data new_data;
set my_data;
if missing(position) then missing_position = 'yes';
else missing_position = 'no';
run;

/*view the resulting dataset with customized missing flags*/
proc print data=new_data;
```

Obs	team	position	points	assists	missing_position
1	A	Guard	14	4	no
2	A	Guard	22	6	no
3	A	Guard	24	9	no
4	A	Forward	13	8	no
5	A	Forward	13	9	no
6	A		10	5	yes
7	B	Guard	24	4	no
8	B	Guard	.	6	no
9	B	Forward	34	2	no
10	B	Forward	15	5	no
11	B	Forward	23	5	no
12	B		10	4	yes

Interpreting Custom Missing Indicators

The resulting `missing_position` column in this third output is significantly more descriptive. It now contains the character value **no** wherever a valid entry exists in the `position` column. Crucially, it flags rows 6 and 12 with the value **yes**, clearly indicating that a missing value was identified for that player's position. This technique is invaluable for data validation and for generating reports where clarity is prioritized over raw numeric indexing. The ability of SAS to pair the basic **MISSING** check with complex IF-ELSE logic provides great flexibility in data preparation.

Further Reference: For detailed technical specifications, syntax variations, and advanced use cases of the SAS **MISSING** function, consult the complete official documentation provided by SAS.

Next Steps in SAS Data Management

Mastering the identification of missing data using the **MISSING** function is merely the first step in effective data management within SAS. Once missing values are identified, the next challenge is deciding on the appropriate strategy for handling them, which may involve deletion, imputation, or specialized modeling techniques.

The following tutorials explain how to perform other common tasks related to data cleansing, transformation, and statistical analysis in SAS: