What is the NOT EQUAL Operator in SAS

Authored by

stats writer

November 19, 2025

The NOT EQUAL operator is a fundamental component of SAS programming, essential for executing sophisticated conditional logic within DATA step and procedure steps. At its core, this operator serves a singular, crucial purpose: determining whether two compared values are distinct or different. This evaluation is critical for filtering records, creating flags, or performing transformations based on whether a specific condition of inequality is met. Understanding the nuances of its syntax and application is key to mastering efficient data manipulation in SAS.

Unlike simple equality checks, the NOT EQUAL comparison returns a true (1) value only when the operands (the values being compared) do not hold the same value. If the values are identical, the comparison returns a false (0) value. This function is most frequently deployed within IF-THEN/ELSE statements, allowing programmers to branch execution flows and generate targeted output datasets based on the disparities found within the raw data.

Syntax and Equivalent Representations of NOT EQUAL

One of the unique aspects of the SAS language is its flexibility in accommodating various syntaxes for the same relational operation, often for the sake of platform portability or user preference. The NOT EQUAL comparison can be expressed using three primary methods, all of which are functionally identical and yield the same result when executed within a SAS program. This redundancy ensures that code written on different operating systems or environments (such as mainframe or PC SAS) remains compatible and readable.

The three accepted symbols and mnemonics for the NOT EQUAL operator are standardized across SAS versions. While the angle brackets represent the traditional mathematical notation for inequality, the mnemonic abbreviation and the caret symbol are favored in certain programming contexts where special characters might be restrictive or less accessible. It is generally recommended to stick to one format within a given project for consistency, although all are valid within the DATA step.

The recognized forms are:

<>: The standard symbolic representation for "not equal to."
ne: The two-letter mnemonic abbreviation derived from "not equal."
^=: The symbolic representation often preferred in environments where the angle bracket combination might cause interpretation issues or for programmers coming from other languages like C or Perl.

For clarity and modern coding standards, many experts prefer the mnemonic form **ne** as it enhances readability, especially for those new to the SAS system, though the functional outcome remains unchanged regardless of the chosen syntax.

Setting the Stage: The Demonstration Dataset

To effectively illustrate the application of the NOT EQUAL operator, we utilize a simple dataset containing metrics for various basketball players. This dataset, named **my_data**, includes crucial information such as team affiliation, player position, and key performance indicators like points scored and assists made. Our objective will be to identify which players had an unequal number of points and assists, generating a new flag variable based on this comparison.

The following code block demonstrates the construction and initial visualization of the sample data. This foundation is essential for understanding how conditional statements operate when applied to real-world statistical data. Pay close attention to the variable definitions--specifically the numeric variables **points** and **assists**, as these will be the focus of our inequality checks in the subsequent examples. The use of the DATA step structure ensures the data is properly instantiated within the SAS environment.

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

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

Viewing the resulting dataset confirms that our input is correctly loaded into the SAS environment, ready for analysis. The initial output (as shown in the image below) displays the raw structure, where rows represent individual players and columns hold their respective statistics. The primary

goal of the subsequent examples is to iterate through these rows and apply the NOT EQUAL operator to the **points** and **assists** columns.

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

Applying Conditional Logic with the 'ne' Operator

The mnemonic **ne** is one of the most commonly used forms for expressing inequality in SAS programming due to its clarity and ease of reading. In this first detailed example, we demonstrate how to use **ne** within an IF-THEN/ELSE structure inside a DATA step to create a new categorical variable. This technique is indispensable for data classification and transformation, forming the backbone of many analytical processes.

The logic flow is straightforward: for every observation (row) in the **my_data** dataset, we compare the value of the **points** variable against the value of the **assists** variable. If these two numeric values are not equal--meaning the condition points ne assists evaluates to true--a new variable, **points_vs_assists**, is assigned the character value 'not equal'. Conversely, if the points and assists values are exactly the same (meaning the condition is false), the operator directs execution to the ELSE clause, and the variable receives the value 'equal'.

This method not only filters or selects data but also creates actionable metadata based on the relational comparison. Below is the code implementation, followed by the resulting dataset structure, which clearly flags rows where the player's score count and assist count differ:

/*create new dataset*/

```
data new_data;
set my_data;
if points ne assists then points_vs_assists = 'not equal';
else points_vs_assists = 'equal';
run;

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

The output dataset confirms the successful application of the NOT EQUAL operator. Observe that the new column, **points_vs_assists**, accurately reflects the comparison outcome. For instance, a player with 14 points and 4 assists receives the 'not equal' designation, whereas a player with 22 points and 22 assists is correctly labeled 'equal'. This process of data flagging is essential for subsequent analyses, such as filtering for only those records where points did not equal assists.

Obs	team	position	points	assists	points_vs_assists
1	A	Guard	14	4	not equal
2	A	Guard	22	22	equal
3	A	Guard	24	9	not equal
4	A	Forward	13	13	equal
5	A	Forward	13	9	not equal
6	A	Forward	10	10	equal
7	B	Guard	24	4	not equal
8	B	Guard	10	6	not equal
9	B	Forward	34	2	not equal
10	B	Forward	15	5	not equal
11	B	Forward	23	23	equal
12	B	Forward	10	4	not equal

Utilizing the '^=' Symbol for Inequality Checks

While **ne** offers high readability, the symbolic representation **^=** (caret followed by equals sign) is a common alternative for expressing the NOT EQUAL relationship in SAS. This notation is particularly familiar to programmers who work with languages derived from C, making it a viable and often interchangeable substitute for **ne** or <>. Functionally, there is no distinction between the use of **ne** and **^=**; they both execute the same comparison logic within the compiler.

To demonstrate this equivalence, we can replicate the exact logic from Example 1, simply substituting the mnemonic **ne** with the symbolic operator **^=**. This exercise reinforces the principle that SAS provides multiple valid ways to achieve the same result, enhancing programmer flexibility. The use of conditional logic remains the same: if the condition points ^= assists is true, the record is flagged; otherwise, the alternative path is taken.

Using the DATA step below, we create an identical dataset, confirming that the choice of NOT EQUAL operator syntax does not affect the logical outcome of the comparison:

```
/*create new dataset*/
data new_data;
set my_data;
if points ^= assists then points_vs_assists = 'not equal';
else points_vs_assists = 'equal';
run;

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

As anticipated, the resulting output matches the output generated in Example 1 exactly. The integrity of the conditional logic is maintained, regardless of whether **ne** or **^=** is used. This consistency is a hallmark of the SAS language design, allowing programmers to choose the syntax that best suits their coding environment or personal style without compromising accuracy.

Obs	team	position	points	assists	points_vs_assists
1	A	Guard	14	4	not equal
2	A	Guard	22	22	equal
3	A	Guard	24	9	not equal
4	A	Forward	13	13	equal
5	A	Forward	13	9	not equal
6	A	Forward	10	10	equal
7	B	Guard	24	4	not equal
8	B	Guard	10	6	not equal
9	B	Forward	34	2	not equal
10	B	Forward	15	5	not equal
11	B	Forward	23	23	equal
12	B	Forward	10	4	not equal

Comparison with Other Relational Operators in SAS

The NOT EQUAL operator is just one tool in the comprehensive set of relational operators provided by SAS. These operators are crucial for defining relationships between variables and facilitating the decision-making processes within the DATA step. Understanding how NOT EQUAL relates to its counterparts helps solidify its role in data manipulation.

Relational operators fundamentally test the relationship between two expressions. Beyond inequality, common SAS operators include **=** (Equal to), **>** (Greater than), **<** (Less than), **>=** (Greater than or equal to), and **<=** (Less than or equal to). Each operator returns a true (1) or false (0) result, which dictates the flow of the program. While equality checks are often used for filtering exact matches, NOT EQUAL is vital for subsetting data where heterogeneity or non-conformance to a standard value is the target.

For example, if we wanted to find players whose points were strictly greater than their assists, we would use the **>** operator. However, if we wanted to find all players whose points and assists were *not* equal, regardless of which value was higher, the NOT EQUAL operator is the most efficient choice. This subtle distinction highlights the power of inequality testing in broad data classification tasks. The choice of operator is dependent entirely on the specific conditional logic requirement of the analytical task.

Handling Missing Values and Data Types in NOT EQUAL Comparisons

When working with relational operators in SAS, special consideration must be given to missing values and data type heterogeneity. SAS treats missing numeric values (represented by a dot, .) as the smallest possible value (negative infinity) for comparison purposes, while missing character values are treated as blanks. This internal handling significantly impacts how the NOT EQUAL comparison evaluates conditions involving incomplete data.

If we compare a numeric variable containing a missing value to a standard numeric value (e.g., 5), the missing value is considered smaller than the standard value. Therefore, in the expression if missing_value ne 5 then..., the condition will be true because a missing value is inherently different from the value 5. Conversely, comparing two missing numeric values (e.g., if . ne . then...) yields a false result, as SAS considers them equal to each other (the smallest possible value).

Furthermore, the NOT EQUAL operator must respect data types. Comparing a numeric variable to a character variable requires SAS to attempt a conversion, which can lead to warnings or errors if the conversion is not clean. It is best practice to ensure that both operands in a NOT EQUAL comparison are of the same data type--either both numeric or both character--to guarantee accurate and predictable results without relying on implicit type conversion by the SAS system. Ensuring data integrity prior to comparison is a critical step in advanced SAS programming.

Advanced Use Cases and Efficiency

The application of the NOT EQUAL operator extends far beyond simple two-variable checks. It is highly effective in filtering out specific codes, handling exceptions, or simplifying complex Boolean conditions. For instance, instead of writing multiple OR conditions to select records where a status variable is A, B, or C, a programmer might use NOT EQUAL to exclude only the unwanted status, D: **if status ne 'D' then output;**.

In terms of code efficiency, using the NOT EQUAL condition appropriately can sometimes simplify code and improve execution speed, particularly when dealing with large datasets where many records conform to the same standard. By only flagging or selecting records that are exceptions (i.e., not equal to the standard), the DATA step avoids unnecessary processing or complex nested IF-THEN statements. Mastering this operator allows for more concise and maintainable SAS code.

The operator's utility also shines in quality assurance and data cleaning routines, where identifying records that deviate from an expected norm is paramount. Any record failing the equality test--for example, if a calculated field does not equal a predefined benchmark--can be isolated using NOT EQUAL for further investigation. This makes **ne**, **^=**, or **<>** a cornerstone of robust statistical programming in SAS.

Summary and Next Steps in SAS Programming

The NOT EQUAL operator is an indispensable element of the SAS programming toolkit, offering multiple syntactic representations (**ne**, **^=**, and **<>**) that provide functional equivalence. Whether used for simple variable comparisons or integrated into complex data validation routines, its ability to quickly ascertain differences between values is fundamental to building effective conditional logic.

As demonstrated through the basketball player examples, mastering this operator is key to performing data segmentation and creating informative indicator variables within the DATA step. Programmers should choose their preferred syntax for consistency but always be aware of the implications regarding missing values and data type matching to ensure reliable results.

To further enhance proficiency in SAS programming, exploring other related topics, such as logical operators (AND, OR), other relational comparisons, and advanced DATA step techniques, is highly recommended.

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