Learn How to Extract Day, Month, and Year from Dates in SAS

Authored by

Mohammed looti

October 31, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Learn How to Extract Day, Month, and Year from Dates in SAS*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=7310

The ability to accurately manipulate and extract components from temporal data is fundamental in statistical programming. In SAS, the **DAY**, **MONTH**, and **YEAR** functions provide streamlined mechanisms to isolate these specific numeric values from a given date variable. Understanding how these functions operate is crucial for tasks ranging from time-series analysis to generating summarized reports based on specific time periods. These functions are typically used within the DATA step to create new variables containing the extracted elements, allowing for enhanced flexibility in data processing and analysis.

While dates might appear straightforward, they are often stored internally in SAS as the number of days since January 1, 1960. This internal numerical representation necessitates the use of specialized functions to convert that raw count back into human-readable components like the day of the month, the month number, or the four-digit year. The following comprehensive examples demonstrate the proper syntax and application of these functions, illustrating how to effectively decompose date information within your statistical workflows. We will begin by establishing a foundational dataset and then proceed to apply the extraction logic, followed by techniques for displaying combined date components using specific format specifications.

Understanding Date Variables in SAS

Before diving into the extraction functions, it is essential to grasp how SAS handles dates. When a date is read into a SAS dataset, it must be assigned a date format (like DATE9. or MMDDYY10.) during the input process. This formatting ensures that the software recognizes the input as temporal data, converting it into the aforementioned internal numeric value. If a date is not correctly formatted, SAS treats it as a standard numeric value, which will lead to incorrect or nonsensical results when attempting to apply date-specific functions like DAY, MONTH, or YEAR.

The utility of separating date components is vast. For instance, analysts often need to group observations by month regardless of the year, or calculate statistics based only on the day of the week (which can be derived using the WEEKDAY function, a close relative of these three). By isolating the components, you transform a single, continuous variable (the date) into three distinct categorical or discrete numerical variables, enabling granular analysis. This process is a common requirement in data preparation and feature engineering tasks, providing a richer structure for subsequent modeling efforts.

The Core Functions: DAY, MONTH, and YEAR Explained

Each of the three functions operates identically, taking one argument: a valid SAS date value or a variable containing a date value. The output is always a standard numeric value.

The **DAY** function returns the day of the month as a number ranging from 1 to 31. For example, if the input date is '14MAR2022'd, the function returns 14. This is particularly useful for identifying specific dates within a month, such as payment due dates or monthly cutoffs. The syntax is straightforward: `day_variable = DAY(input_date_variable);`.

The **MONTH** function returns the month of the year as a number ranging from 1 (January) to 12 (December). Using the same input, '14MAR2022'd, the function returns 3. This numeric output facilitates easy calculation of quarterly sums or monthly averages. Similarly, the syntax is `month_variable = MONTH(input_date_variable);`.

Finally, the **YEAR** function returns the four-digit year. For '14MAR2022'd, the function returns 2022. This result is crucial for time-series analysis requiring yearly aggregation or cohort definition. The syntax follows the same pattern: `year_variable = YEAR(input_date_variable);`. These functions are highly efficient and are the preferred method for numerical date decomposition in SAS.

Example 1: Setting up the Sample Dataset

To illustrate the practical application of these functions, we must first define a sample dataset. This dataset contains a single date variable, `birth_date`, representing the birth date for several individuals. Notice the use of the `format birth_date date9.` statement, which ensures that SAS reads the input dates correctly in the DDMMMYY format, recognizing them as temporal values rather than simple strings or numbers. This step is non-negotiable when dealing with date manipulation.

The following code employs the DATA step combined with the `DATALINES` statement to hardcode the input data directly into the session. This method is common for quick examples and testing code logic before applying it to larger, external data sources. The dates are listed sequentially, spanning several years, which will allow us to observe the correct extraction of day, month, and year values simultaneously.

Suppose we have the following dataset in SAS that shows the birth date for seven individuals:

```
/*create dataset*/
data original_data;
format birth_date date9.;
input birth_date :date9.;
datalines;
01JAN2021
22FEB2022
14MAR2022
29MAY2022
14OCT2023
01NOV2024
26DEC2025
;
run;

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

Upon execution, the PROC PRINT procedure displays the newly created dataset, confirming that the dates were read and formatted correctly, ready for the component extraction process outlined in the next step.

Obs	birth_date
1	01JAN2021
2	22FEB2022
3	14MAR2022
4	29MAY2022
5	14OCT2023
6	01NOV2024
7	26DEC2025

Applying Functions to Extract Granular Date Components (Example 1 Continued)

The core objective here is to transform the single date column, `birth_date`, into three distinct numeric variables: `day`, `month`, and `year`. This is achieved by iterating through the `original_data`

dataset using the `SET` statement within a new DATA step, and applying the corresponding functions to the source variable.

We create the new dataset, `new_data`, which inherits all variables from `original_data`. Within this step, we define the three new variables, explicitly calling **DAY**, **MONTH**, and **YEAR** on the `birth_date` variable. It is important to remember that these new variables (day, month, year) are standard numeric variables; they do not retain any special date properties, but rather hold the extracted integer values, which can then be used in arithmetic operations or grouping statements.

The following code snippet demonstrates this simultaneous extraction and variable creation process. We then use PROC PRINT again to display the contents of `new_data`, verifying that the new columns contain the precise numerical components derived from the original date.

```
/*create new dataset*/
data new_data;
set original_data;
day = DAY(birth_date);
month = MONTH(birth_date);
year = YEAR(birth_date);
run;

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

As expected, the output confirms the successful decomposition of the original date. The three new variables show the day, month (as a number 1-12), and year (as a four-digit number) corresponding precisely to the **birth_date** variable in each observation. This separated data is now ready for deep statistical analysis or reporting that requires time-period segmentation.

Obs	birth_date	day	month	year
1	01JAN2021	1	1	2021
2	22FEB2022	22	2	2022
3	14MAR2022	14	3	2022
4	29MAY2022	29	5	2022
5	14OCT2023	14	10	2023
6	01NOV2024	1	11	2024
7	26DEC2025	26	12	2025

Advanced Extraction Techniques: Focusing on Month and Year Formatting (Example 2)

While the **MONTH** and **YEAR** functions are excellent for extracting numerical components for calculation, sometimes the requirement is simply to display the date in a combined month-year format for reporting purposes, without necessarily creating new numeric variables for aggregation. In this scenario, utilizing specific SAS format specifications is generally more efficient and stylistically cleaner than combining the output of two separate functions.

We can create a new variable, `month_year`, and assign it the value of the original `birth_date` variable. Crucially, since the `month_year` variable now holds the raw SAS date value (the number of days since 1960), we can apply a specific date format to control its display. The `MMYYN6.` format is designed to display the two-digit month and the two-digit year, separated by a specified character (here, a hyphen or a slash, depending on the system default, but often implied by the 'N' in the format name), within a six-character width.

This approach avoids the need to run the DAY, MONTH, and YEAR functions, making the code more concise when only display modification is needed. The following code demonstrates how to create a new variable that displays just the month and year of a date variable in SAS using the `MMYYN6.` format:

```
/*create new dataset*/
data new_data;
set original_data;
month_year = birth_date;
format month_year mmyyn6.;
```

```
run;

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

After running this code, the PROC PRINT output clearly shows that the new variable **month_year** contains only the month and year of the **birth_date** variable, formatted neatly as MM/YY.

Obs	birth_date	month_year
1	01JAN2021	012021
2	22FEB2022	022022
3	14MAR2022	032022
4	29MAY2022	052022
5	14OCT2023	102023
6	01NOV2024	112024
7	26DEC2025	122025

Controlling Date Display Order with SAS Formats

The choice of date format in SAS is highly flexible and depends entirely on the required output standard. While `MMYYN6.` displays Month followed by Year (MM/YY), international or specific organizational standards might require the year to precede the month (YY/MM). Fortunately, SAS provides an analogous format specification to handle this inversion instantly, without requiring any complex recoding or manipulation of the underlying date values.

If you need the year to appear before the month in the output, you simply substitute `MMYYN6.` with `YYMMN6.` in the DATA step. This change affects only the display layer; the numeric value stored in the `month_year` variable remains the internal SAS date count, ensuring continuity if the variable were to be used in further date calculations later in the program. This highlights the power of SAS formats: they are presentation tools that overlay the raw data, offering powerful control over how data is viewed.

The following revised code demonstrates the use of the `yymmn6.` format to achieve the Year/Month display order, maintaining the same underlying date dataset structure and calculation logic.

```
/*create new dataset*/
data new_data;
set original_data;
month_year = birth_date;
format month_year yymmn6.;
run;

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

This flexibility demonstrates that SAS offers two primary paths for date manipulation: using **DAY**, **MONTH**, and **YEAR** functions when numerical separation is required for calculations or grouping, and using specialized date formats like `MMYYN6.` or `YYMMN6.` when the goal is purely to control the presentation of combined components.

Conclusion and Best Practices

Mastering date manipulation is essential for effective data analysis in SAS. The **DAY**, **MONTH**, and **YEAR** functions provide analysts with robust tools for extracting fundamental numerical components from any valid date variable, facilitating detailed temporal breakdowns required for reporting and statistical modeling. These functions reliably return integer values that can be used directly in conditional logic (e.g., IF statements) or in procedures like PROC MEANS for aggregation.

When choosing between using these functions and applying specific formats, the rule of thumb is simple: use the functions if you need the component as a numeric variable for further calculations (like summing total sales by year), but use formats (e.g., `MMYYN6.`) if you only need to change the visual representation of the date without altering its underlying numerical structure or creating new variables. Always ensure your input date variables are correctly formatted using statements like `format variable date9.` during the initial DATA step to prevent common errors related to date recognition.

Additional Resources

The following tutorials explain how to perform other common tasks in SAS, building upon the foundational knowledge of date handling presented here:

How to Calculate Age from Birth Date in SAS

How to Use the WEEKDAY Function in SAS
How to Convert Date to Numeric in SAS