When ANY Function Will Just NOT Do

Richann Watson
Karl Miller

ABSTRACT

Have you ever been working on a task and wondered whether there might be a SAS® function that could save you some time? Let alone, one that might be able to do the work for you? Data review and validation tasks can be time-consuming efforts. Any gain in efficiency is highly beneficial, especially if you can achieve a standard level where the data itself can drive parts of the process. The ANY and NOT functions can help alleviate some of the manual work in many tasks such as data review of variable values, data compliance, data formats, and derivation or validation of a variable's data type. The list goes on. In this poster, we cover the functions and their details and use them in an example of handling date and time data and mapping it to ISO 8601 date and time formats.

INTRODUCTION

Moving forward it has become clear that CDISC standards are the key role in data submission to the FDA. The cleaner the data, the more compliant the data, the better off the process will be and the quicker the approval for the drug. One of the key areas to assist in this new process, gaining efficiency in working with the data and getting from SDTM to ADaM as quickly as possible while ensuring traceability, is to utilize robust programming and letting the data drive as much work as possible. Through working the data, or letting the data work the program, and utilization of SAS functions not only can programs be utilized across studies but can also help in identification of issues with the data. Well established, data-based programming can get you to where you want to be faster and with more accuracy while reducing manual error. The ANY and NOT functions are just a few of the many functions available so we encourage you to explore and expand your knowledge through these functions in SAS.

ANY OR NOT FUNCTIONS

Programming through the use of functions can lead to reducing the assumptions based on what the data should be like along with any other inconsistencies, reducing extended time in debugging programs when the resulting data is not according to specifications and/or controlled terminology. We have found a few functions that can be useful in working with the data.

Based solely on the data, the ANY functions return the first position in which the indicated/assigned character is found while the NOT functions return the first position in which the indicated character is not found. In Table 1 below you can find a few examples in the use of these functions.

Function Search Criteria	ANY Function	NOT Function
alpha-numeric character	ANYALNUM(*string, <start>*)	NOTALNUM(*string, <start>*)
alphabetic character	ANYALPHA(*string, <start>*)	NOTALPHA(*string, <start>*)
digit	ANYDIGIT(*string, <start>*)	NOTDIGIT(*string, <start>*)
character that is valid as the first character of a SAS variable*	ANYFIRST(*string, <start>*)	NOTFIRST(*string, <start>*)
lowercase letter	ANYLOWER(*string, <start>*)	NOTLOWER(*string, <start>*)
character that is valid in a SAS variable*	ANYNAME(*string, <start>*)	NOTNAME(*string, <start>*)
punctuation character	ANYPUNCT(*string, <start>*)	NOTPUNCT(*string, <start>*)
white-space character: blank, horizontal and vertical tab, carriage return, line feed, and form feed	ANYSPACE(*string, <start>*)	NOTSPACE(*string, <start>*)
uppercase letter	ANYUPPER(*string, <start>*)	NOTUPPER(*string, <start>*)

* SAS variable name under VALIDVARNAME=V7

Table 1. Examples of criteria based ANY/NOT functions

APPLICATIONS

For both the ANY and NOT functions, *string* is a required parameter. String can either be a character constant, variable or an expression. For both types of functions, *start* is optional. When used, how it is specified or populated will determine the direction in which the search is performed. Specific cases would be:

- Start > 0, search is from left to right
- Start < 0, search is from right to left
- Start < negative length of string, search starts at the end of the string (e.g., length of the string is 15 and the start has a value of -20, the search would start at the end of the string.)

The result of the functions is numeric, based on the corresponding variable's data values where the search criterion can be found. For the specific cases, ANY and NOT functions will return the value of zero (0) when one of the following is true:

- Search character is not found
- Start > length of the string (e.g., length of the string is 15 and the start has a value of 20)
- Start = 0

SCENARIO 1

SDTM Data Example

Rather than manual review of the data, whether through programming to get all unique values by using a PROC SORT NODUPKEY or other methods, you can eliminate the majority of common values that can cause mapping and other data programming issues. One case in particular, collected lab results, are an area where this commonly occurs when processing the data in configuration to SDTM standards.

For the LB domain, the lab result is collected 'as is' and then standardized to a common unit of measure. Prior to programming the standardized result, you need to determine if LBORRES has any alphabetic characters included to decide on the type of processing that should and/or can be used in converting to LBSTRESC and/or LBSTRESN.

The use of ANYALPHA can be used in this instance to focus on the specific cases where the results contain an alphanumeric character. This will help in reducing the number of records that you need to review and ensure that you are including the collected results through the assignment of the correct values into the correct corresponding variables.

```
anyalpha(lborres)
```

If the ANYALPHA returns a value > 0, then a list of those values can be provided to the appropriate person to aid in the review of the data to determine, if there are data issues or to determine if they are truly character values. Table 2 provides an example of how ANYALPHA could be used.

LBORRES	ANYALPHA	Handling
1.4	0	Convert to standardized unit and populate LBSTRESC and LBSTRESN
TRACE	1	No conversion required. Copy to LBSTRESC.
CANCELLED	1	Data should be queried or fixed so that LBSTAT = 'NOT DONE' and LBREASND = 'CANCELLED'

Table 2. Example of ANYALPHA used in SDTM to aid in data review

ADaM Data Example

Continuing to work with the laboratory data, but rather on the analysis side of programming a study, we can use the functions to determine if AVALC contains a partial numeric result, and if so is it an integer or float. This technique can be useful when specifying the datatype for each parameter in a define.xml. Through the use of the ANYALPHA and ANYPUNCT functions you can prepare the data for potential use in analysis needs.

```
not(anyalpha(AVALC))

anypunct(AVALC)
```

Based on the segments of code, ANYALPHA will yield a 0 if there are no alphabetic characters but for this instance we are wanting to process only the numeric results so we take the ***not*** of the ANYALPHA to yield 1. It might be tempting to use NOTALPHA for this, but it is also very important to keep in mind what is actually being searched by the functions. Since NOTALPHA will return the position of the first non-alphabetic character and due to the case that some character results can contain both alphabetic and numeric characters, the use of NOTALPHA would yield a non-zero value for results that are alphanumeric and not strictly numeric.

Table 3 is a sample table of some values and their respective results of using the ANY/NOT functions that illustrates why NOTALPHA or NOTDIGIT would not be ideal in this scenario.

AVALC	ANYALPHA	NOTALPHA	ANYPUNCT	NOTDIGIT
TESTING	1	8	0	1
<1.0	0	1	1	1
1.2	0	1	2	2
354.12	0	1	4	4
	0	1	0	1
___123	0	1	1	1
124!	0	1	4	4
47821	0	1	0	6
1TEST	2	1	0	2

Table 3. Sample illustration of several ANY and NOT functions

You might ask “Why not just use NOTALPHA function and if it returns a value of ‘1’, then the result is numeric?” As you can see in the above table, on the rows in red font, it shows that a blank value returns a value of ‘1’ and we don’t want to process this record. Furthermore, the third to last and the last rows start with a number but then have either punctuation or an alpha character and the NOTALPHA stops processing after it finds the first non-alpha character, which in these cases, is the first character. Again, you would not want to process these records.

SCENARIO 2

For scenario 2, we take a look at data manipulation or computation which will be based on the incoming data values and used to generate the final variable values. One common case in working with CDISC standards would be visit mapping. Whether cycles, weeks, or days, the functions can create a standard set of code that can be used time and time again, with minimal maintenance to correctly assign the correct values.

Week Only Example

In working across multiple data vendors you can have visit information that is in a variety of formats, but what is consistent is that there is usually a number that is preceded by characters, a type of punctuation and/or white space.

For this example, we illustrate the use of the ANYDIGIT function and when it is best to use the ANY or NOT functions versus a more traditional approach. The more traditional approaches would be along the lines to use a SUBSTR and INDEXC functions, or simply to use a COMPRESS function.

```
visind = input(substr(visit,indexc(visit, '123456790')), best.);
viscomp = input(compress(visit,,'dk'), best.);
```

Aside from these more traditional approaches, another would be to use SUBSTR and ANYDIGIT functions.

```
visany = input(substr(visit, anydigit(visit)), best.);
```

All three approaches work and yield the same result as illustrated in Table 4.

VISIT	VISIND	VISCOMP	VISANY
w4	4	4	4
w 4	4	4	4
W-4	4	4	4
WK4	4	4	4
Wk: 4	4	4	4
Week 4	4	4	4
WEEK:4	4	4	4

Table 4. Week Only Example

In this scenario, for week only, the traditional approach of using a COMPRESS function (i.e., viscomp) would be ideal to use but what if you would have the week and day information and you need to split them into their respective variables? Our next example demonstrates how to handle this when the visit involves both 'Week and Day'.

Week and Day Example

You can have visit information that is in a variety of formats, but what is consistent in these formats is the fact that you have a number that is preceded by characters, punctuation, and/or white space. This number represents the week. This week number is then followed by more characters, punctuation, and/or white space, with the last character in the string being a number which represents the day.

We want to be able to extract the week as a variable and the day as a separate variable. As indicated, the data can be in a variety of formats so how do we do this without using a series of '***if-then-else***' statements along with having to constantly check the data and update the code for any new scenarios?

In the previous example, we showed how to extract the number portion of a visit when the raw data comes in a variety of formats. We now expand upon that example to illustrate how we can extract different portions of a visit, mainly the week and day information:

a. Find the location of the first number when searching from the left.

```
firstnumloc = anydigit(visit);
```

b. Find the location of the first alpha character when searching from the left starting the search at the position of the first number.

```
secalploc = anyalpha(visit, firstnumloc);
```

c. Find the location of the first number when searching from the right

```
lastnum = anydigit(visit, -length(visit));
```

d. Extract the week portion using the location of the first number when searching from the left (a.) and the location of the first alpha character after the first number when searching from the left

(b.).

```
WEEK = input(substr(visit, firstnumloc, secalploc-firstnumloc), best.);
```

e. Extract the day portion using the location of the first number when searching from the right (c.).

```
DAY = input(substr(visit, lastnum), best.);
```

Based on the above steps, you have now extracted the week and day information from the visit string with only 5 lines of code. This will be dynamic enough to handle a variety of scenarios and you don't have to manually check for new formats and update the program. Table 5 illustrates the values at each step and what the final Week and Day values are.

VISIT	FIRSTNUMLOC	SECALPLOC	LASTNUM	WEEK	DAY
W4D1	2	3	4	4	1
w 4 d 1	3	5	7	4	1
W4 D1	2	4	5	4	1
w-4 d-1	3	5	7	4	1
wk4 d1	3	5	6	4	1
wk4 dy1	3	5	7	4	1
wk 4 dy 1	4	6	9	4	1
WK: 4 DY: 1	5	7	11	4	1
WEEK 4 DAY 1	6	8	12	4	1
week:4 day:1	6	8	12	4	1

Table 5. Week and Day Example

SCENARIO 3

One of the requirements of CDISC is that for SDTM domains the date/time variables are to be in ISO 8601 format. This format allows for a variation in how the date/time is captured. For example, if the raw data has date only for some records and date and time for other records and partial dates in other records, in order to capture the necessary information, the raw data ends up having several different variables that house the collected date and/or time. In some instances the raw data will capture each date component and each time component as individual variables. The ISO 8601 format will allow these different date/time scenarios to be captured in one variable.

Date and Time Example

For this example in working with dates and times, the data are captured as individual components in character format and looking at the data you see that there is no consistent way in which the data is captured (see Table 6). Again, rather than writing a series of '***if-then-else***' statements that would need to be maintained if data comes in with a value that you did not previously account for, how do you handle this to get the date/time in the correct format per CDISC standards?

MON	DAY	YEAR	HR	MIN	SEC
01	01	2016	01	30	45
01	01	2016	01	30	--
01	01	2016	01	30	UN
01	01	2016	01	--	UN
01	01	2016	UN	30	
01	01	2016	UN	UK	NA
01	--	2016	UN	UK	NA
NA	--	2016	UN	UK	NA

MON	DAY	YEAR	HR	MIN	SEC
UK	01	2016	o1	30	
NA	01	2016	UK	30	
--	01	2016	NA	30	
01	--	2016	O1	30	
01	NA	2016	01	30	
UK	UK	2016	01	30	
01	01	UNK	01	30	
0I	01	NA		30	
--	UK	UNK	01	30	45
--	UK	UNK		NA	
11	19	UNK	UK	NA	--

Table 6. Date and Time in Raw Data

Given that each component is character and anything can be entered, each component needs to be converted to a valid value that can be used to create the ISO 8601 date.

a. The NOTDIGIT can help determine which component has values other than numbers. If the component has only numeric characters then a new variable is created that is a copy of the original value. However, if the component has something other than a number the component would be considered missing and would be denoted with a single dash.

```
if not(notdigit(&dttmvar.)) then _&dttmvar. = &dttmvar.;
 else _&dttmvar. = '-';
```

b. Combine the date components to create a date variable that is in ISO 8601 format as well as combine the time components to create the time variable that is in ISO 8601 format. At this point you may assume you are done, but a true ISO 8601 format will only display up through the last non-missing component and as the date and time variables stand, they can all be missing because we used '-' to denote the missing component.

```
isodt = catx("-", _year, _mon, _day);
isotm = catx(":", _hr, _min, _sec);
```

c. Using NOTPUNCT determine if the time is completely missing. If the time is completely missing it will be in the form of '-:-:-'. If there is at least one non-missing time component, then the NOTPUNCT will return the position of that non-missing component.

```
if notpunct(strip(isotm)) > 0 then …
```

d. Now that you have fixed the time so that it only contains non-missing times (i.e., it is not '-:-:-'), then it can be combined with the date to form the ISO 8601 datetime variable. However, you are still not finished. What happens if there is no time and you had a partial date?

```
newdttm = catx("T", isodt, _isotm);
```

e. Using ANYALPHA will determine if there is a time portion to the ISO 8601 variable. If there is a time, then the ISO 8601 datetime variable is ready. However, if ANYALPHA returns 0 then it is an indication that there is no time (i.e., step c showed that the time was of the form '-:-:-') so the date needs to be processed to find the last non-missing portion of the date.

```
if anyalpha(strip(newdttm)) > 0 then NEWDTC = newdttm;
```

f. Using NOTPUNCT in searching from left to right the first time to see if there are any numbers in the date string. If found, then continue using NOTPUNCT searching from right to left will find the last numeric character in the string.

```
… if notpunct(strip(newdttm)) > 0 then
    NEWDTC = substr(newdttm, 1, notpunct(strip(newdttm),
                    -length(newdttm)));
```

After execution of the steps, in the indicated order, you have a new date/time variable that is in ISO 8601 format and you were not required to go through the data to find all the possible ways that each date component and each time component was entered.

For the sake of saving space, note that Table 7 only shows the necessary variables to illustrate the concepts above and the complete code can be found in the Appendix.

MON	DAY	YEAR	HR	MIN	SEC	…	ISODT	_ISOTM	…	NEWDTC
01	01	2016	01	30	45		2016-01-01	01:30:45		2016-01-01T01:30:45
01	01	2016	01	30	--		2016-01-01	01:30		2016-01-01T01:30
01	01	2016	01	30	UN		2016-01-01	01:30		2016-01-01T01:30
01	01	2016	01	--	UN		2016-01-01	01		2016-01-01T01
01	01	2016	UN	30			2016-01-01	-:30		2016-01-01T-:30
01	01	2016	UN	UK	NA		2016-01-01			2016-01-01
01	--	2016	UN	UK	NA		2016-01--			2016-01
NA	--	2016	UN	UK	NA		2016----			2016
UK	01	2016	o1	30			2016---01	-:30		2016---01T-:30
NA	01	2016	UK	30			2016---01	-:30		2016---01T-:30
--	01	2016	NA	30			2016---01	-:30		2016---01T-:30
01	--	2016	O1	30			2016-01--	-:30		2016-01--T-:30
01	NA	2016	01	30			2016-01--	01:30		2016-01--T01:30
UK	UK	2016	01	30			2016----	01:30		2016----T01:30
01	01	UNK	01	30			--01-01	01:30		--01-01T01:30
0I	01	NA		30			----01	-:30		----01T-:30
--	UK	UNK	01	30	45		-----	01:30:45		-----T01:30:45
--	UK	UNK		NA			-----			
11	19	UNK	UK	NA	--		--11-19			--11-19

Table 7. Date and Time in Raw Data

CONCLUSION

As you can see in the examples that were covered the use and implementation of functions into your programming can bring in some simple code, but beware that if not used correctly it can yield some unexpected results. Not only are these functions simple yet powerful, but can also perform soundly across multiple data sources and data types that you may be working with on a single study or even an entire compound. In considering the correct function for establishing robust code for use in programming, it is always advised that you are not only aware of the selected functions use but also the application of that knowledge to correctly structure its use to meet your needs. With all factors and needs considered, the selection of the most appropriate function can prove to be highly beneficial in efficiency and successful in your programming.

REFERENCES

Dictionary of SAS Functions and CALL Routines. SAS Institute, Available at http://support.sas.com/documentation/cdl/en/lefunctionsref/67960/HTML/default/viewer.htm.

RECOMMENDED READING

- *SAS® Functions by Example* by Ron Cody

APPENDIX

```
data dttm2;
       set dttm;

       /* convert individual date/time components to valid values */
       /* for ISO 8601 datetime variable                          */
       %macro cnvt(dttmvar=);
         if not(notdigit(&dttmvar.)) then _&dttmvar. = &dttmvar.;
         else _&dttmvar. = '-';
       %mend cnvt;

       %cnvt(dttmvar=year)
       %cnvt(dttmvar=mon)
       %cnvt(dttmvar=day)
       %cnvt(dttmvar=hr)
       %cnvt(dttmvar=min)
       %cnvt(dttmvar=sec)

       /* use new variables to be build ISO 8601 date time */
       isodt = catx("-", _year, _mon, _day);
       isotm = catx(":", _hr, _min, _sec);

       /* if time is nothing but '-' and ':' then default to blank */
       /* if it there is so numeric portion of the time then keep  */
       /* up throught he last time element that had a numeric part */
       if notpunct(strip(isotm)) > 0 then
           _isotm = substr(isotm, 1, notpunct(strip(isotm), -length(isotm)));
       else _isotm = ' ';

       /* combine time with date to build ISO datetime */
       newdttm = catx("T", isodt, _isotm);

       /* if there is no time portion then keep only up */
       /* through last numeric portion of date          */
       if anyalpha(strip(newdttm)) > 0 then NEWDTC = newdttm;
       else if notpunct(strip(newdttm)) > 0 then
          NEWDTC = substr(newdttm, 1, notpunct(strip(newdttm), -length(newdttm)));
       else NEWDTC = ' ';
run;
```