

Cody's Collection of Popular SAS[®] Programming Tasks and How to Tackle Them



Ron Cody

Contents

List of Programs	ix
About This Book.....	xv
About The Author.....	xix
Acknowledgments	xxi
Chapter 1 Tasks Involving Conversion: Character to Numeric, Specific Values to Missing, and Changing Case	1
Introduction.....	1
Task: Converting character values to numeric values.....	2
Task: Converting character values to numeric values using a macro	3
Task: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set.....	5
Task: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set	7
Task: Changing all character values to either uppercase, lowercase, or proper case.....	8
Task: Reading a numeric value that contains units such as Lbs. or Kgs. in the value	9
Task: Solving part of the previous task using a Perl regular expression.....	10
Conclusion	11
Chapter 2 Grouping Data	13
Introduction.....	13
Task: Grouping values using IF-THEN-ELSE statements.....	13
Task: Grouping values using user-defined formats	14

Task: Creating groups using PROC RANK	15
Conclusion	19
Chapter 3 Summarizing Data	21
Introduction	21
Task: Using PROC MEANS to create a data set containing summary information	22
Task: Computing the mean of a variable broken down by values of another variable: Using a BY variable	23
Task: Computing the mean of a variable broken down by values of another variable: Using a CLASS statement	24
Task: Have PROC MEANS name the variables in the output data set automatically (the AUTONAME option)	25
Task: Creating multiple output data sets from PROC MEANS, each with a different combination of CLASS variables	26
Task: Combining summary information (a single mean) with detail data: Using a conditional SET statement.....	29
Task: Combining summary information (a single mean) with detail data: Using PROC SQL.....	31
Task: Combining summary information (a single mean) with detail data: Using PROC SQL without using PROC MEANS	32
Task: Combining summary information (a single mean) with detail data: Using a macro variable.....	33
Task: Combining summary data with detail data—for each category of a BY variable	34
Conclusion	36
Chapter 4 Combining and Updating SAS Data Sets	37
Introduction.....	37
Task: Concatenating two SAS data sets—Using a SET statement	38
Task: Concatenating two SAS data sets—Using PROC APPEND	40
Task: Concatenating two SAS data sets with character variables of different lengths.....	40
Task: Concatenating two SAS data sets that contain character variables of different lengths and controlling the length of the character variables.....	42

Task: Developing a macro to concatenate two SAS data sets that contain character variables of different lengths.....	43
Task: Updating a SAS data set using a transaction data set.....	47
Task: Using a MODIFY statement to update a master file from a transaction file	50
Task: Updating several variables using a transaction file created with an INPUT method called named input.....	50
Task: Matching names from two SAS data sets where the names may not be spelled the same (fuzzy merge).....	53
Conclusion	56
Chapter 5 Creating Formats from SAS Data Sets	57
Introduction.....	57
Task: Using a SAS data set to create a format (by creating a control data set).....	57
Task: Adding new format values to an existing format	62
Conclusion	63
Chapter 6 Table Lookup Techniques	65
Introduction.....	65
Task: Performing a one-way table lookup using a MERGE statement	65
Task: Performing a one-way table lookup using user-defined informats	67
Task: Creating an INFORMAT using a control data set	69
Task: Performing a one-way table lookup using a temporary array	70
Task: Performing a two-way table lookup using a temporary array	71
Conclusion	73
Chapter 7 Restructuring (Transposing) SAS Data Sets.....	75
Introduction.....	75
Task: Converting a data set with one observation per subject into one with multiple observations per subject (using a DATA step)	76
Task: Converting a data set with one observation per subject into one with multiple observations per subject (using PROC TRANSPOSE).....	77
Task: Converting a data set with multiple observations per subject into one with one observation per subject (using a DATA step).....	79

Task: Converting a data set with multiple observations per subject into one with one observation per subject (using PROC TRANSPOSE)	80
Conclusion	81
Chapter 8 Tasks Involving Dates	83
Introduction	83
Task: Computing a person’s age, given his or her date of birth	83
Task: Computing a SAS date given a month, day, and year (even if the day value is missing)	84
Conclusion	85
Chapter 9 Data Cleaning Tasks	87
Introduction	87
Task: Looking for possible data errors using a given range	88
Task: Demonstrating a macro to report on outliers using fixed ranges	89
Task: Demonstrating a macro that performs automatic outlier detection	92
How the macro works	94
Conclusion	96
Chapter 10 Reading Data with User-Defined INFORMATS	97
Introduction	97
Task: Reading a combination of character and numeric data	97
Conclusion	100
Chapter 11 Tasks Involving Multiple Observations per Subject	101
Introduction	101
Task: Using PROC SORT to detect duplicate BY values or duplicate observations (records)	102
Task: Extracting the first and last observation in a BY group	106
Task: Detecting duplicate BY values using a DATA step	108
Task: Identifying observations with exactly 'n' observations per subject	109
Task: Computing differences between observations (for each subject)	110

Task: Computing the difference between the first and last observation for each subject	112
Conclusion	114
Chapter 12 Miscellaneous Tasks	115
Introduction	116
Task: Determining the number of observations in a SAS data set (using the NOBS= SET option)	116
Task: Determining the number of observations in a SAS data set and assigning the value to a macro variable	117
Task: Determining the number of observations in a SAS data set (using library tables)	118
Task: Determining the number of observations in a SAS data set (using SAS functions)	119
Task: Counting the number of a specific response in a list of variables	120
Task: Computing a moving average	122
Task: Presenting a macro to compute a moving average	124
Task: Replacing the first eight digits of a credit card number with asterisks	126
Task: Sorting within an observation (using the ORDINAL function)	127
Task: Sorting within an observation (using CALL SORTN)	128
Task: Computing the average of the 'n' highest scores	129
Task: Extracting the first and last name (and possibly a middle name) from a variable containing the first and last name (and possibly a middle name) in a single variable	130
Index	133

Chapter 1: Tasks Involving Conversion: Character to Numeric, Specific Values to Missing, and Changing Case

Introduction	1
Task: Converting character values to numeric values	2
Task: Converting character values to numeric values using a macro.....	3
Task: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set.....	5
Task: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set.....	7
Task: Changing all character values to either uppercase, lowercase, or proper case	8
Task: Reading a numeric value that contains units such as Lbs. or Kgs. in the value.....	9
Task: Solving part of the previous task using a Perl regular expression	10
Conclusion	11

Introduction

This chapter contains programs to perform character-to-numeric conversion, one of the most common tasks you will face as a SAS programmer. You will see a sample program as well as a useful macro that accomplishes this goal.

Another task that you will probably face is converting a specific numeric value such as 999 or a specific text value such as 'NA' to a SAS missing value.

In this chapter, you will also see how to convert every character variable to a specific case, such as uppercase.

The last task in this chapter demonstrates how to read data values that contain units, such as 100Lbs. or 50Kgs. and create a numeric variable with all of the values using the same units.

Task: Converting character values to numeric values

Keywords

Character-to-numeric conversion

Swap and Drop

How many times have you been given a SAS data set with variables such as Height or Weight but, instead of being numeric variables, they are stored as character? The following example describes how to convert these character variables to numeric variables, maintaining the original variable names.

For this example, you start out with a SAS data set called Char_values. Here is a listing:

Age	Weight	Gender	DOB
23	150	M	10/21/1983
67	220	M	09/12/2001
77	101	F	05/06/1977

If you run PROC CONTENTS on this data set, you see that Age and Weight are character variables. The following program performs the conversion:

Program 1.1: Converting character values to numeric values

```
*Converting character values to numeric;

data Num_Values;
  set Char_Values(rename=(Age = C_Age
                          Weight = C_Weight));
  Age = input(C_Age,best12.);
  Weight = input(C_Weight,best12.);
  drop C_;;
run;
```

The “trick” here is to rename the variables as they are read from the input data set. This allows you to use the original variable names for the resulting numeric variables. The character-to-numeric conversion is performed using the INPUT function. You don’t have to worry if the INFORMAT used in the INPUT function represents more digits than you need—unlike an INPUT statement, you can never read past the end of a character value when using the INPUT function.

Notice the variable list on the DROP statement `c_*`: The colon acts as a wildcard suffix. `c_*` represents all variables that begin with the characters C followed by an underscore.

The resulting data set has exactly the same variables as the original data set except the two variables Age and Weight are now numeric. A partial listing from PROC CONTENTS confirms this:

Alphabetic List of Variables and Attributes				
#	Variable	Type	Len	Format
3	Age	Num	8	
2	DOB	Num	8	MMDDYY10.
1	Gender	Char	1	
4	Weight	Num	8	

Task: Converting character values to numeric values using a macro

Keywords

Character-to-numeric conversion

Conversion macro

Because character-to-numeric conversion is required in so many situations, this chapter offers you a macro that performs the conversion automatically. As in the previous program, the resulting data set uses the same variable names as in the original data set that contains the character variables. Here is the macro, followed by an explanation:

Program 1.2: Presenting a macro to perform character-to-numeric conversion

```
*Macro to convert selected character variables to
numeric variables;
%macro char_to_num(In_dsn=, /*Name of the input data set*/
                  Out_dsn=, /*Name of the output data set*/
                  Var_list= /*List of character variables that you
                             want to convert from character to
                             numeric, separated by spaces*/);

/*Check for null var list */
%if &var_list ne %then %do;
/*Count the number of variables in the list */
%let n=%sysfunc(countw(&var_list));
data &Out_dsn;
  set &In_dsn(rename=(
  %do i = 1 %to &n;
```

```

/* break up list into variable names */
  %let Var = %scan(&Var_list,&i);
/*Rename each variable name to C_ variable name */
  &Var = C_&Var
%end;
));

%do i = 1 %to &n;
  %let Var = %scan(&Var_list,&i);
  &Var = input(C_&Var,best12.);
%end;
drop C_;;
run;
%end;
%mend char_to_num;

```

The calling arguments in this macro are the names of the input and output data sets and a list of the variables that you wish to convert from character to numeric. You enter the names of each variable in this list, separated by spaces.

The first task of the macro is to rename each of the original variable names by appending the prefix `C_` to each of the names. To determine how many variable names there are in `&Var_list`, you use the `COUNTW` function. This function computes the number of words in a string. To obtain each of the variable names, you use the `%SCAN` macro function. This function works in the same way as the regular non-macro `SCAN` function. The first argument is the list of variable names. The second argument specifies which “word” you want in the string. The macro uses a `%DO` loop to extract each of the individual variable names.

The next `%DO` loop performs the character-to-numeric conversion using the `INPUT` function. Notice that the first argument of the `INPUT` function is the original variable name with the `C_` prefix added. Finally, a `DROP` statement deletes all of the `C_` variables.

To test the macro, you can use the original data set `Char_values` and enter `Age` and `Weight` as the argument of `Var_List`. Here is the code:

Program 1.3: Testing the character-to-numeric conversion macro

```

*Test the macro;
%char_to_num(In_dsn=char_values, Out_dsn=Num_values,
             Var_list=Age Weight)

```

After you run the macro, the output data set (`Num_values`) is identical to the one created by the previous program.

Task: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set

Keywords

Numeric variables

`_numeric_`

Array

You will find numerous occasions where you need to perform an operation on all numeric (or character) variables in a SAS data set. For example, you may have a SAS data set where specific values, such as 999 or 9999, were used to represent a missing value. In the character domain, you may want to convert all character values to uppercase or convert a specific value such as 'NA' to a SAS missing value. The approach to all of these tasks is the same. You create an array of all numeric or character variables. Once you do this, you can then use a DO loop to perform any operation you desire on all of the variables in the array.

This first example converts a value of 999 to a SAS missing value for all the numeric variables in data set Demographic.

A listing of data set Demographic is shown here:

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
1	70	999	76	04NOV1955	Male	NA
2	26	160	62	08APR1955	NA	NA
3	71	195	71	20JUL1955	male	na
4	40	132	74	08JAN1955	Male	Republican
5	999	181	62	15AUG1951	Female	Democrat
6	62	71	52	24JAN1950	Male	democrat
7	24	136	72	26NOV1950	Female	democrat
8	5	174	71	08NOV1950	Female	democrat
9	5	172	47	28DEC1951	Male	Democrat
10	94	173	999	06MAY1953	Male	republican
11	99	170	63	27FEB1950	na	NA
12	10	133	63	18MAR1954	Male	democrat
13	6	131	60	26MAR1951	Female	republican

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
14	999	140	79	01OCT1950	NA	na
15	999	124	999	12OCT1950	NA	na
16	44	194	72	31DEC1952	Female	republican
17	62	196	68	09MAR1951	Female	democrat
18	57	133	72	15SEP1951	Female	Democrat
19	45	137	86	16NOV1951	NA	Republican
20	90	170	80	01OCT1951	Female	Republican

You will use this data set for several of the tasks in this chapter. For this example, notice that there are several values of 999 for the variables Score, Weight, and Heart_Rate.

Here is the code that performs the conversion:

Program 1.4: Converting a specific value such as 999 to a missing value for all numeric variables in a SAS data set

```
*Converting a specific value such as 999 to a missing value for
all numeric variables in a SAS data set;

data Num_missing;
  set Demographic;
  array Nums[*] _numeric_;
  do i = 1 to dim(Nums);
    if Nums[i] = 999 then Nums[i] = .;
  end;
  drop i;
run;
```

The key to this program, as well as several programs to follow, is to create an array using the keyword `_NUMERIC_`. When used in a DATA step, `_NUMERIC_` represents all the numeric variables that have been defined up to that point in the DATA step. Since the ARRAY statement follows the SET statement, the Nums array contains all of the numeric variables in data set Demographic (Subj, Score, Heart_Rate, and DOB). To make this important point clear, had you placed the ARRAY statement before the SET statement, the array Nums would not contain any variables.

You certainly do not want to have to count all the numeric variables in a large data set. Therefore, you use an asterisk in the brackets following the array name. When you do this, SAS will count the number of variables for you. But, what value do you use in the DO loop? You can use the DIM (dimension) function to determine how many variables are in the array. Your work is almost finished. All you need to do now is to check for values of 999 and convert them to a SAS numeric missing value. Don't forget to drop the DO loop counter.

The first five observations in data set Num_missing are shown next, to demonstrate that the program worked as expected:

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
1	70	.	76	04NOV1955	Male	NA
2	26	160	62	08APR1955	NA	NA
3	71	195	71	20JUL1955	male	na
4	40	132	74	08JAN1955	Male	Republican
5	.	181	62	15AUG1951	Female	Democrat

Task: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set

Keywords

Character variables

character_Array

This task is similar to the previous task. The difference is that you want to convert a specified character value to a SAS character missing value. All you need to do is use the SAS keyword `_CHARACTER_` to create an array of all character variables. Here is the program:

Program 1.5: Converting a specific value such as 'NA' to a missing value for all character variables in a SAS data set

```
*Converting a specific value such as "NA" to a missing value for all
character variables in a SAS data set;
data Char_missing;
  set Demographic;
  array Chars[*] _character_;
  do i = 1 to dim(Chars);
    if Chars[i] in ('NA' 'na') then Chars[i] = ' ';
  end;
  drop i;
run;
```

Array Chars contains all the character variables in data set Demographic (in this case, Gender and Party). As in the previous task, the DIM function returns the number of variables in the array. To make the program more general, it looks for uppercase or lowercase values of 'NA'. Here is a listing of the first five observations in data set Char_missing:

Subj	Score	Weight	Heart_Rate	DOB	Gender	Party
1	70	999	76	04NOV1955	Male	
2	26	160	62	08APR1955		
3	71	195	71	20JUL1955	male	
4	40	132	74	08JAN1955	Male	Republican
5	999	181	62	15AUG1951	Female	Democrat

Task: Changing all character values to either uppercase, lowercase, or proper case

Keywords

Uppercase

Lowercase

Proper case

character

In a similar manner to the previous program, you can use an array of all your character variables to convert them all to a unified case: uppercase, lowercase, or proper case. Please refer to the previous program if you would like an explanation of this program. As you can see, this program is converting all the character values in the Demographic data set to uppercase. The two other functions that convert character values to lowercase or proper case are LOWCASE and PROPCASE, respectively. Here is the program:

Program 1.6: Changing case for all character variables in a SAS data set

```
*Converting all character values to uppercase (or lower- or proper-
case);
data Upper;
  set Demographic;
  array Chars[*] _character_;
  do i = 1 to dim(Chars);
    Chars[i] = upcase(Chars[i]);
  end;
  drop i;
run;
```

If the character variables you are dealing with represent names and addresses, after you have converted all the values to a consistent case, you may want to take the additional step and use the COMPBL function to convert all multiple blanks to a single blank, to help standardize the names and addresses.

Task: Reading a numeric value that contains units such as Lbs. or Kgs. in the value

Keywords

Character-to-numeric conversion

Removing units from a value

Extracting digits from a string

COMPRESS function

SCAN function

Data set Units contains a character variable called Weight that includes units such as Lbs. and Kgs. (pounds and kilograms). To add insult to injury, the variable Height also contains units and it is expressed in feet and inches (sometimes the inches value is missing (when the inches value is zero)). A listing of data set Units is shown here:

Subj	Weight	Height
001	80kgs	5ft 3in
002	190lbs	6' 1"
003	70KG.	5ft 11in
004	177LbS.	5' 11"
005	100kgs	6ft

Notice that the Weight units are not always in the same case and some of the units end in periods. For Height, the abbreviation 'ft' or 'in' is used; sometimes a single quote and double quote represent feet and inches.

You would like to create two new variables (Weight_Lbs and Height_Inches) that are numeric variables and are equal to the weight in pounds and the height in inches, respectively. Here is the program:

Program 1.7: Reading data values that contain units

```
*Reading data values that contain units;
data No_Units;
  set Units;
  Weight_Lbs = input(compress(Weight,,'kd'),12.);
  if findc(Weight,'k','i') then Weight_lbs = Weight_lbs*2.2;
  Height = compress(Height,,'kds');
  Feet = input(scan(Height,1,' '),12.);
  Inches = input(scan(Height,2,' '),12.);
```

```

if missing(Inches) then Inches = 0;
Height_Inches = 12*Feet + Inches;
drop Feet Inches;
run;

```

You start by extracting the digits from Weight using the COMPRESS function with the modifiers 'kd' (keep digits). It is important to include two commas following the first argument of the COMPRESS function so that the function interprets 'kd' as modifiers and not the second argument to the COMPRESS function that is used to list the characters you want to compress from a string. Since the result of the COMPRESS function is a character value, you use the INPUT function to perform the character-to-numeric conversion. All you need to do is test the original variable (Weight) to see if it contains a 'K' in uppercase or lowercase. Use the FINDC function with the 'i' modifier (ignore case) to do this. If you find a 'K', you multiply by 2.2 to convert from kilograms to pounds.

The Height variable presents more of a challenge. You first use the COMPRESS function with three modifiers, 'kds' (keep digits and space characters). The variable Height now contains two sets of digits (or only a single digit if there are zero inches) and can use the SCAN function to extract the feet and inch values. The SCAN function returns a missing value for Inches if Height only contains a single number (feet). You can now add 12 times the feet plus the number of inches to obtain the height in inches. Here is the listing of the data set No_Units:

Subj	Weight	Height	Weight_Lbs	Height_Inches
001	80kgs	5 3	176	63
002	190lbs	6 1	190	73
003	70KG.	5 11	154	71
004	177LbS.	5 11	177	71
005	100kgs	6	220	72

Solving this task without the COMPRESS and SCAN functions would certainly be a challenge—with these functions, it's a snap.

Task: Solving part of the previous task using a Perl regular expression

Keywords

Removing units from a value

Extracting digits from a string

Perl regular expression

My younger son, who is a wizard at programming, suggested I solve this problem using a Perl regular expression. This solution is not simpler than the previous solution, but it demonstrates the versatility of regular expressions.

You start by using PRXPARSE to compile the regular expression:

```
/^(\d+) (\D)/
```

This regex (this is what Perl programmers call regular expressions) is looking for one or more digits followed by a non-digit. The ^ in the beginning of the expression says to start the search at the beginning of the string. The digit and non-digit values will be placed in capture buffers because each of these expressions is in a set of parentheses. You use the PRXMATCH function to search for the pattern of a number followed by a non-number. The PRXPOSN function extracts the values in each of the capture buffers. The INPUT function performs the character-to-numeric conversion as in the previous task.

If the value in the second capture buffer is a 'K', you perform the kilogram to pound conversion.

Program 1.8: Using a Perl regular expression to extract the digit and units part of a character value

```
*Solution using Perl Regular expressions;
data No_Units;
  set Units(drop=Height);
  if n_ = 1 then do;
    Regex = "/^(\d+) (\D)/";
    re = prxparse(Regex);
  end;
  retain re;
  if prxmatch(re,Weight) then do;
    Weight_Lbs = input(prxposn(re,1,Weight),8.);
    Units = prxposn(re,2,Weight);
    if upcase(Units) = 'K' then Weight_Lbs = Weight_Lbs*2.2;
  end;
  keep Subj Weight Weight_Lbs;
run;
```

The resulting data set contains values for Weight_Lbs that are identical to the values in the previous task.

Conclusion

It is quite likely that you will need to perform one or more of the tasks described in this chapter on a regular basis. Since the character-to-numeric conversion is one of the most common tasks, you may choose to store the conversion macro in your macro library.

Also keep in mind that using the special keywords `_NUMERIC_` and `_CHARACTER_` to define an array can save you immense time when you need to perform an operation on all character or numeric variables in a data set.

About The Author



Ron Cody, EdD, is a retired professor from the Robert Wood Johnson Medical School who now works as a private consultant and a national instructor for SAS. A SAS user since 1977, Ron's extensive knowledge and innovative style have made him a popular presenter at local, regional, and national SAS conferences. He has authored or co-authored numerous books.

Learn more about this author by visiting his author page at support.sas.com/cody. There you can download free chapters, access example code and data, read the latest reviews, get updates, and more.

RON CODY PROVIDES THE RECIPES FOR SAS® PROGRAMMING SUCCESS

Purchase [Cody's Collection of Popular SAS® Programming Tasks and How to Tackle Them](#).

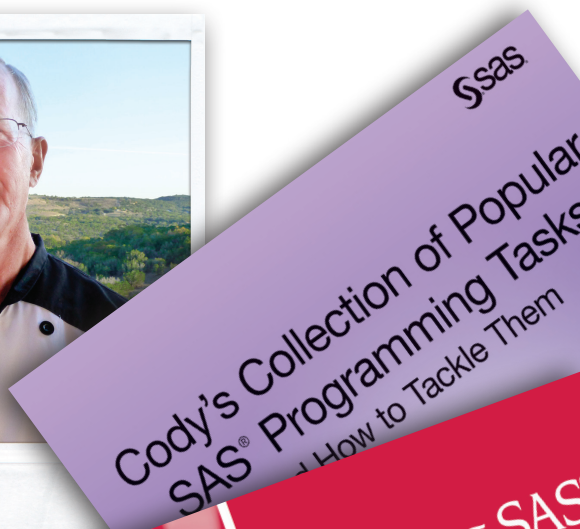
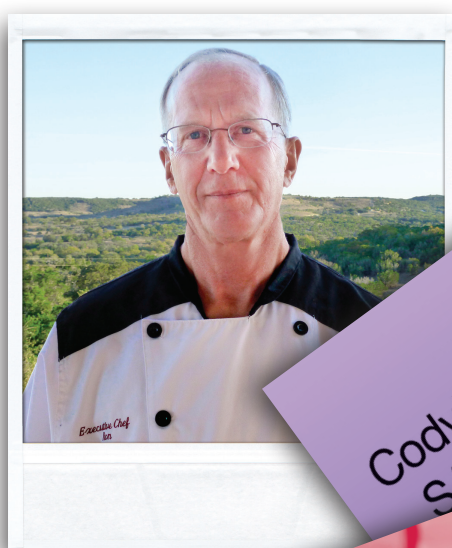
Learn more about Ron Cody, read free chapters from his books, and access example code and data on his [author page](#).

Browse our [full catalog](#) to find additional books that are just right for you.

Subscribe to our [monthly e-newsletter](#) to get the latest on new books, documentation, and tips—delivered to you.

Browse and search free [SAS documentation](#) sorted by release and by product.

Email us: sasbook@sas.com
Call: 800-727-3228



SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brands and product names are trademarks of their respective owners.

Cody, Ron; Cody's Collection of Popular SAS® Programming Tasks and How to Tackle Them. Copyright © 2012, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED. For additional SAS resources, visit support.sas.com/bookstore.