Paper 3978-2019

Standardizing Text in your Data

Guihong Chen, TCF Bank

ABSTRACT

SAS® offers a variety of useful tools to manipulate character data. These are very handy when data received may not be particularly clean. For example, a character string may contain inconsistent or unwanted special character or formats that cause reconciliation or merging job to fail. This paper provides an example using customer data to discuss several functions to efficiently standardize text variables. Undesirable information can be removed by leveraging the three arguments of the COMPRESS function. Undesirable blanks can be removed by the TRIM, TRIMN or STRIP functions. Inconsistent formats or special characters can be standardized with a consistent character using the TRANWRD, UPCASE, LOWCASE, and PROPCASE functions. However, there may a need to add or reorder a substring. The concatenation operator (||) or functions CAT, CATT, CATS, CATX, SUBSTR, or SCAN can also be used to clean and modify a dataset. The intended audience for this presentation is beginner to intermediate SAS users with good knowledge of Base SAS.

INTRODUCTION

Text variables in a SAS data set may not be as clean as desired. These variables may contain inconsistent formats or unwanted special characters, which makes it difficult to identify duplicate observations or join with other data sets by these text variables. SAS offers a variety of useful tools to manipulate character data. This paper discusses several functions used to standardize text variables.

Table 1 shows a small data set with customers' basic information: name, phone number, and address with inconsistent formats for every field. To make this information useful, standardizing the data is required for further analysis or reconciliation with another file.

Table 1: Raw Data Set

ID	Name	Phone	Zip	City	Address
1	Davis, Alan	727-528-0507	ZIP#33781	PINELLAS PARK	198 Highland Rd.
2	Ryan Barnette	9313635630	38478-9273	PULASKI	1305 S MAIN ST
3	Alan Davis	(727)528-0507	33781	Pinellas Park	198 Highland Road

The table above can be generated by using the code below:

```
   data Example;
infile datalines delimiter='||';
length ID 8. Name $ 30 Phone $20 Zip $10  City $30 Address $100 ;
input ID  NAME $ phone $ Zip $ City $ Address  ;
datalines;
1|| Davis, Alan || 727-528-0507 || ZIP#?33781 || PINELLAS PARK || 198
Highland Rd.
2|| Ryan Barnette ||9313635630 || 38478-9273 ||PULASKI || 1305 S MAIN ST
3|| Alan Davis || (727)528-0507 || 33781 || Pinellas Park || 198 Highland
Road
run;
```

1. NAME-- REORDER A SUBSTRING

There are two different formats of names in the data set. One format, as seen on line one shows last name before first name, using a comma and a blank as the separators. The other format, as seen on line two and line three has first name before the last name using a blank as the separator. This section shows how to standardize the names into the format as seen in the second and third customers.

First, define the first name and last name of any given label by using SCAN function. SCAN function returns the nth word from a character string. The full version of SCAN function has four arguments as SCAN (string, count, delimiter, modifiers). The first two arguments (string and count) are required while the last two arguments (delimiter and modifiers) are optional and may be used as needed.

Since the positions of the first names and last names vary with two different formats, different count parameters must be used in the SCAN function to find the first names and last names in these two formats. The INDEX function can be used to distinguish these two formats. The INDEX function searches a character expression for a string of characters and returns the position of the string's first character for the first occurrence of the string. If no such string is found, the index function returns 0. Searching for a comma (“,”), the INDEX function will return a positive integer for the first customer and zero for the second and third customer. Then one can define first names and last names by using the code below:

```
   data Name0;SET Example;
if INDEX(Name,",")>0 then do;
      Last_Name=SCAN(Name,1);
      First_Name=SCAN(Name,2);
      End;
else do;
      Last_Name=SCAN(Name,2);
      First_Name=SCAN(Name,1);
      End;
RUN;
```

In this example, the basic version of the **SCAN (string, count, delimiter, modifiers)** function is used with the two required arguments:

- The first argument (**string**) can be a character constant, variable, or expression;
- The second argument (**count**) is a nonzero integer value that specifies the number of the word in the character string that you want SCAN to select.

Since the delimiter is a blank or comma in this example, one does not need to define the third and fourth arguments as blank and comma are the default delimiters.

Next, concatenate First_Name and Last_Name in the desired order. Concatenation operator (||) or concatenation functions can be used as shown in the code below:

```
   data Name;set Name0;
Name2=First_Name||" "||Last_Name;
Name3=TRIM(First_Name)||" "||TRIM(Last_Name);
Name4=TRIMN(First_Name)||" "||TRIMN(Last_Name);
Name5=STRIP(First_Name)||" "||STRIP(Last_Name);
Name6=COMPRESS(First_Name)||" "||COMPRESS(Last_Name);
Name7=TRIM(First_Name)||" "||COMPRESS(Last_Name);
Name_CATX=CATX(" ",First_Name,Last_Name);
Name_CAT=CAT(First_Name," ", Last_Name);
Name_CATT=CATT(First_Name," ", Last_Name);
Name_CATS=CATS(First_Name," ", Last_Name);

KEEP name: First_Name Last_Name;
run;
```

The results are summarized below in Table 2. Some ways are not generating the desirable results. The desirable results are marked in green.

Table 2: Standardizing Names

ID	1	2	3
Name	Davis, Alan	Ryan Barnette	Alan Davis
Last_Name	Davis	Barnette	Davis
First_Name	Alan	Ryan	Alan
Name2	Alan Davis	Ryan	Alan Davis
Name3	Alan Davis	Ryan Barnette	Alan Davis
Name4	Alan Davis	Ryan Barnette	Alan Davis
Name5	Alan Davis	Ryan Barnette	Alan Davis
Name6	Alan Davis	Ryan Barnette	Alan Davis
Name7	Alan Davis	Ryan Barnette	Alan Davis
Name_CATX	Alan Davis	Ryan Barnette	Alan Davis
Name_CAT	Alan Davis	Ryan	Alan Davis
Name_CATT	AlanDavis	RyanBarnette	AlanDavis
Name_CATS	AlanDavis	RyanBarnette	AlanDavis

The code above generates the following 10 different variables for full names:

- Name2 uses the concatenation operator (||) alone but does not generate the desired name string. The reason is that First_Name and Last_Name have the same length as Name variable which is 30 bytes. Each value of First_Name and Last_Name ends with several trailing blanks to occupy the 30 bytes space.
- Name3 to Name7 use four different functions to remove trailing blanks: TRIM, TRIMN, STRIP, COMPRESS. Each of these functions can be applied to First_Name or Last_Name before concatenation to create the desired name string. The differences in these four functions rely on the different treatment of leading blanks and inside blanks. TRIM function removes trailing blanks only while TRIMN is the same as TRIM function except that it returns no character if a string is blank. STRIP has an additional power over TRIMN by removing the leading blanks. COMPRESS is even more powerful than STRIP in that it can remove the blanks between first and last name. Note that in the desired name string, only one separator: a blank between First_Name and Last_Name is used.
- Name_CATX generates the identical desired name by concatenating First_Name and

Last_Name while removing leading and trailing blanks and inserting the desired blank as a separator.

- Name_CAT is the same as Name2 as CAT function is equivalent to concatenation the operator (||), which did not generate a desirable name.
- Name_CATT is not generating the exact same desired name because the blank is removed from between the first and last names. The CATT function removes the trailing blanks of each component but also returns no character for a blank string component so the blank separating the first and last names is removed.
- Name_CATS is not generating the exact same desired name either as the blank is removed from the separator. The CATS function removes the leading and trailing blanks of each component but also removes the blank separator as CATS returns no character for a blank string component.

2. PHONE--ADD SEPARATORS INSIDE A STRING

Phone numbers are not consistent in the example above as some customers' numbers use hyphens ("-") or parentheses "()" as separators while one customer's number does not have any separators. It is possible to standardize all the phone numbers to that of the first customer. Again, the INDEX function is used to distinguish the different phone formats as shown below:

```
   data Phone;SET Example;
if INDEX(phone,"-")=0 then Phone2=SUBSTR(Phone,1,3)||"-
"||SUBSTR(Phone,4,3)||"-"||SUBSTR(phone,7);
else if INDEX(phone,"(")>0 then Phone2=SUBSTR(Phone,2,3)||"-
"||SUBSTR(Phone,6,3)||"-"||SUBSTR(phone,10);
else phone2=phone;
run;
```

Since the phone numbers have the same number of digits and the separators are inserted in the fixed position of the number, the SUBSTR function can be used to find the positions to add the separators. The **SUBSTR** (**string, position, length**) function has three arguments:

- The first argument (**string**) is the text to extract from.
- The second argument (**position**) is the starting position from which to extract the string.
- The third argument (**length**) is optional and stands for the length of the substring to extract from the first argument. If the third argument is not supplied, the default third argument is the length of the string from the position provided by the second argument to the end of the string.

The output of the standardized phone number is illustrated below in the last row of Table 3. The first line of phone numbers shows the original formats for reference.

Table 3: Standardizing Phone Numbers

ID	1	2	3
Name	Davis, Alan	Ryan Barnette	Alan Davis
Phone	727-528-0507	9313635630	(727)528-0507
Phone2	727-528-0507	931-363-5630	727-528-0507

3. ZIP--REMOVE SPECIAL TEXTS FROM A STRING

The zip codes are not consistent in that the first customer's zip code has some unwanted letters and a special character (#) and the third customer's zip code has more than 5 digits. The zip codes are standardized by using the COMPRESS function to remove special characters from the original string in the code below:

```
  data Zip;SET Example;
Zip2=SUBSTR(COMPRESS(zip,,"pa"),1,5);
run;
```

The output of the standardized zip codes is illustrated below in the last row of Table 4. The first line of zip codes shows the original formats for reference.

Table 4: Standardizing Zip Codes to 5 Digits

ID	1	2	3
Name	Davis, Alan	Ryan Barnette	Alan Davis
Zip	ZIP#33781	38478-9273	33781
Zip2	33781	38478	33781

The COMPRESS function with one argument can be used to remove the trailing blanks and standardize the name in the first section. The COMPRESS function can have three arguments as **COMPRESS (source, chars, modifiers)**:

- The first argument (**source**) specifies the source string to be compressed.
- The second argument (**chars**) specifies a string or list of characters to be removed.
- The third argument (**modifiers**) is optional and used to modify the list of the second argument.

The zip code of the first customer has unwanted letters and pound sign (#). To remove the unwanted letters, add "a" to the third argument. To remove pound sign or other punctuation marks such as &*@^:,=!-.|+?, add "p" to the third argument. The SUBSTR function is used outside the COMPRESS function to make sure only the first 5 digits are used for the zip codes.

4. CITY—CHANGE THE CASE FOR A STRING

In the example, city names are not consistent in that the first and second customers' city names are upper cases while the third customer's city name is lower case with the first letter capitalized. SAS provides three ways to change the case of the letters in a string:

- UPCASE (argument): Converts all the letters in the argument to uppercase
- LOWCASE (argument): Converts all the letters in the argument to lowercase
- PROPCASE (argument, delimiter): Converts all the letters in the argument to proper case. The default delimiters for the second argument are blank, forward slash, hyphen, open parenthesis, period, and tab.

The PROPCASE function provides a more readable way to capitalize a city name, as shown below:

```
   data City;SET Example;
City2=PROPCASE(City);
run;
```

The output of the standardized city names is illustrated in the last row of Table 5. The first line of city names shows the original formats for reference.

Table 5: Standardizing Cities

ID	1	2	3
Name	Davis, Alan	Ryan Barnette	Alan Davis
City	PINELLAS PARK	PULASKI	Pinellas Park
City2	Pinellas Park	Pulaski	Pinellas Park

5. ADDRESS—REPLACE A SUBSTRING WITH ANOTHER SUBSTRING

When it comes to writing an address, it is acceptable to abbreviate words such as Street, Avenue, Place, Road, Square, Boulevard, Terrace, Drive, Court, Crescent, Station and Building. However, addresses appear inconsistent when some addresses abbreviate words while others do not. In the example, the first and second customers abbreviate Road and Street while the third customer does not. It is best to standardize the address using the same abbreviation rules. TRANWRD function can be used to replace a substring with another user-defined substring as in the code below:

```
   data Address;SET Example;
Address2=PROPCASE(TRANWRD(UPCASE(Address),"ROAD","Rd."));
Address3=PROPCASE(TRANWRD(UPCASE(Address2),"ST","St."));
run;
```

TRANWRD (source, target, replacement) function has three required arguments:

- The first argument (**source**) specifies a character constant, variable, or expression that you want to translate.
- The second argument (**target**) specifies a character constant, variable, or expression that is searched for in source. The length for target must be greater than zero.
- The third argument (**replacement**) specifies a character constant, variable, or expression that replaces the target. When the replacement string has a length of zero, TRANWRD uses a single blank instead.

The output of the standardized addresses is illustrated below in the last row of Table 6. The first line of addresses shows the original formats for reference.

Table 6: Standardizing Addresses

ID	1	2	3
Name	Davis, Alan	Ryan Barnette	Alan Davis
Address	198 Highland Rd.	1305 S MAIN ST	198 Highland Road
Address2	198 Highland Rd.	1305 S Main St	198 Highland Rd.
Address3	198 Highland Rd.	1305 S Main St.	198 Highland Rd.

6. COMBINE ALL STANDARDIZING STEPS IN ONE DATA SET

The data comparison before and after all the standardization steps is shown below. By standardizing all the character variables in the data set, it is easy to find that the first and the third customer have the same information. Lastly, remove the duplicate customer information in the data. The comparison results are summarized in Table 7 and clean data after removing duplicate customer information is illustrated in Table 8.

Table 7: Comparison before and after the standardization

ID		Name	Phone	Zip	City	Address
1	Before	Davis, Alan	727-528-0507	ZIP#33781	PINELLAS	198 Highland Rd.
	After	Alan Davis	727-528-0507	33781	Pinellas Park	198 Highland Rd.
2	Before	Ryan Barnette	9313635630	38478-9273	PULASKI	1305 S MAIN ST
	After	Ryan Barnette	931-363-5630	38478	Pulaski	1305 S Main St.
3	Before	Alan Davis	(727)528-0507	33781	Pinellas Park	198 Highland Road
	After	Alan Davis	727-528-0507	33781	Pinellas Park	198 Highland Rd.

Table 8: Clean Data after standardization and de-dupe process

Name2	Phone2	Zip2	City2	Address3
Alan Davis	727-528-0507	33781	Pinellas Park	198 Highland Rd.
Ryan Barnette	931-363-5630	38478	Pulaski	1305 S Main St.

The code below can be used to generate the results and remove the duplicate customer information:

```
data Example2;set Example;
if INDEX(Name,",")>0 then Last_Name=scan(Name,1);else Last_Name=scan(Name,2);
if INDEX(Name,",")>0 then First_Name=scan(Name,2);else
First_Name=scan(Name,1);
Name2=CATX(" ",First_Name,Last_Name);

if INDEX(phone,"-")=0 then Phone2=SUBSTR(Phone,1,3)||"-
"||SUBSTR(Phone,4,3)||"-"||SUBSTR(phone,7);
else if INDEX(phone,"(")>0 then Phone2=SUBSTR(Phone,2,3)||"-
"||SUBSTR(Phone,6,3)||"-"||SUBSTR(phone,10);
else phone2=phone;
Zip2=SUBSTR(COMPRESS(zip,,"pa"),1,5);
City2=PROPCASE(City);
Address2=PROPCASE(TRANWRD(UPCASE(Address),"ROAD","Rd."));
Address3=PROPCASE(TRANWRD(UPCASE(Address2),"ST","St."));
run;
```

```
  proc sql;
select distinct name2, phone2, Zip2, City2, Address3 from Example2;quit;
```

CONCLUSION

Character variables must be handled with care, as they may come from different data sources and with different formats and lengths, making the data difficult to analyze. Furthermore, duplicate information may not be easily identified and data joining could be difficult. Being familiar with the character functions mentioned in this paper can help use data more efficiently.

REFERENCES

- SAS Online Documentation,
  http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000212246.htm
  http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000215027.htm
  http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000214639.htm
  http://support.sas.com/documentation/cdl/en/imlug/66845/HTML/default/viewer.htm#imlug_langref_sect455.htm
- Virginia Chen, “Dealing with Blanks: Leading, Trailing, In-Between” SUG2009,
  https://www.lexjansen.com/nesug/nesug09/cc/CC25.pdf

ACKNOWLEDGMENTS

I am grateful to Duke Owen for his encouragement, guidance and always interesting correspondence. My colleagues in TCF bank Brad Olson and Chris Docken also helped to produce what you find here.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Guihong Chen
TCF Bank
612-202-0550
guihongc@gmail.com