
Cleaning Dirty Data With Just A Handful of SAS Functions

Ben Cochran

The Bedford Group

bencochran@nc.rr.com

An Affiliate Member of the SAS Alliance

Contents

1. Leading Zero Blaster
 2. LENGTH Function
 3. LEFT / ZIPSTATE
 4. TRANSLATE
 5. MODTE
 6. PROPCASE
 7. COMPRESS
 8. ANYUPPER
-

1. Leading Zero Blaster

A certain organization has a character variable that contains leading zeros followed by some number. They want to create a new variable without the leading zeros. Use the INDEXC and SUBSTR functions to do this.

```
data leading_0;
  input Number $;
  Non_0 = indexc(Number, '123456789');
  New_Number = substr(Number, Non_0);
cards;
0123
007_0K
00033Y
proc print;
run;
```

Obs	Number	Non_0	New_
1	0123	2	123
2	007_0K	3	7_0K
3	00033Y	4	33Y

2. Length Function

A certain dataset has a City_State variable that contains both the city and state.

Write a DATA step to separate the State from the City.

The challenge is the city value has several embedded blanks and varies in length.

Notice that the values for State occupy only 2 spaces at the end of the string. There are NO commas in this variable (City_State).

VIEWTABLE: Work.City	
	city_state
1	King and Queen Court House VA
2	Saint Mary of the Woods IN
3	West Palm Beach FL
4	Outer Banks NC

Length Function

Step 1. Use the LENGTH function to determine the Length of the **value** of the string.

Step 2. Grab the rightmost 'word' and put it in STATE.

Step 3. Put the rest of the string in CITY.

```
data city_state;  
  set city;  
  length state $ 5;  
  len=length(city_state);  
  state = scan(city_state, -1);  
  city=substr(city_state, 1, len - 3);  
run;
```

The **LENGTH** function returns the length of the value of **city_state** to a variable named **len**.

VIEWTABLE: Work.City_state				
	city_state	state	len	city
1	King and Queen Court House VA	VA	29	King and Queen Court House
2	Saint Mary of the Woods IN	IN	26	Saint Mary of the Woods
3	West Palm Beach FL	FL	18	West Palm Beach
4	Outer Banks NC	NC	14	Outer Banks

3. Data Cleaning with 3 Functions

patient_id	name	county	state	zip_code	emergency
A01101	Smith, Jean	Orange	NC	27515-2688	Y
A99126	Moore, Ronald	Wake	NC	27511-2414	N
B031073	Adams, Beth	Wake	NC	27606-4010	Y
B001324	Polinski, Gus	Durham	NC	27705-2102	N

Here is what we need to do:

Compare the **zip code** with the value of state and make sure the zip code is in the correct state.

Q. What function in particular is needed to do this?

A. ZIPSTATE.

However, the ZIPSTATE function only works with the first 5 digits of the zip code.

How can we access only the first 5 digits of zip code?

cleaning

Data Cleaning

The SUBSTR function retrieves the first 5 digits from zip_code.

```
data zip_check (keep= name county zip5);  
  set patient;  
  length zip5 $5;  
  zip5=substr(zip_code, 1, 5);  
run;  
proc print data = zip_check;  
run;
```

Partial PROC PRINT output. Examine ZIP5. What happened?

Obs	name	county	zip5
1	Smith, Jean	Orange	2751
2	Moore, Ronald	Wake	2751
3	Adams, Beth	Wake	2760
4	Polinski, Gus	Durham	2770
5	Pegg, Bill Jr	Durham	2770
6	Fox, Mary P	Durham	3770
7	Perez, Rose	Dallas	7503

How can we fix this?

left →

7

Note: There is actually a leading blank in ZIP_CODE, and consequently in ZIP5 also.

The LEFT Function

There are several methods that could be used here. We will use the LEFT function to left align a character variable. Here is the syntax and how it works.

The typical form of the LEFT function is:

LEFT (argument)

where argument is a character variable or expression.

Suppose the variable **zip_code** is a character variable with a length of 11 and has the following value:

zip_code \$ 11
27607-1234

Illustrate the use of the LEFT function:

ex . **x = left (zip_code);**

zip_code \$ 11	x \$ 11
27607-1234	27607-1234

The value of 'x' is left aligned while the value of 'zip_code' remains right aligned.

left □

The LEFT Function

Use the LEFT function to eliminate leading blanks in a character field before doing any comparisons. Then use the ZIPSTATE function to see if retrieve the state in which the zip code is found.

```
data good bad(keep= name county state state_check zip5);
  set patient;
  length zip5 $5;
  zip5=substr(left(zip_code), 1, 5);
  state_check = zipstate(zip5);
  if left(state_check) ne left(state) then output bad;
  else output good;
run;
proc print data=bad;
run;
```

Notice 'zip5' and 'state_check'.

Obs	name	county	state	zip5	state_check
1	Fox, Mary P	Durham	NC	37705	TN
2	Fisher, Jo	New Hanover	NC	29891	SC
3	Mathis, Curt	New Hanover	NC	39891	

There are three patients whose zip code does NOT match their state.



3. Translate Function

Earlier it was discovered that there was a problem with some of the street numbers in the **Street_Address** field. Some of the street numbers actually contain letters.

Task: Write a DATA step to fix this problem.

VIEWTABLE: Sasuser.A_patient			
	patient_id	patient	street_address
1	A01101	Ms. Jean Smith	4 Comer St.
2	A99126	Mr. Ronald Moore	130 Market St.
3	B031073	Ms. Beth Adams	442l Glenwood Ave.
4	B001324	Mr. Gus Polinski	18o Cannon Dr.
5	A03121	Mr. Bill Pegg Jr.	10L Cannon Dr.
6	B991401	Mrs. Mary P. Fox	2lO Ear Ave.
7	A021313	Miss Rose Perez	301 Lucky Dr.

First, isolate the street numbers and convert all the letters to numbers (numbers).

```
data fix_it (keep=numbers new);  
  set sasuser.A_patient;  
  numbers = scan(street_address, 1, '');  
  new=translate(numbers,'0011', 'OoLI');  
run;
```

VIEWTABLE: Work.Fix_it		
	numbers	new
1	4	4
2	130	130
3	442l	4421
4	18o	180
5	10L	101
6	2lO	210
7	301	301

Street_Address Numbers, and New all have a length of 21.

The TRANSLATE function is used here to convert any one of these letters: 'Oo' to the digit '0' (zero), and any of these letters: 'LI' to the digit '1'.

Translate Function

Step 2. Modify the DATA step to take the corrected street numbers and use them to rebuild the variable **Street_Address**.

```
data fix_it (drop=numbers new );
  set sasuser.A_patient;
  numbers = scan(street_address, 1, ' ');
  new=translate(numbers,'0011','OoLI');
  → space = index(street_address, ' ');
  street_address = trim(new) !! substr(street_address, space);
run;
```

Notice that NEW is trimmed in the DATA step. Where did the space after the street numbers in the address come from?

	patient_id	patient	street_address	space
1	A01101	Ms. Jean Smith	4 Comer St.	2
2	A99126	Mr. Ronald Moore	130 Market St.	4
3	B031073	Ms. Beth Adams	4421 Glenwood Ave.	5
4	B001324	Mr. Gus Polinski	180 Cannon Dr.	4
5	A03121	Mr. Bill Pegg Jr.	101 Cannon Dr.	4
6	B991401	Mrs. Mary P. Fox	210 Ear Ave.	4
7	A021313	Miss Rose Perez	301 Lucky Dr.	4

4. The ATTRN Function

The **ATTRN** function returns information about a **numeric** attribute of an open SAS data set. The typical syntax is:

```
ATTRN (dsid, attribute-name);
```

Selected values of **ATTRIBUTE-NAME** are: any, modte, nobs, nlobs, nvars, etc.

The values of **RC** are dependent on the attribute-name. For the **ANY** attribute:

- -1 means the data set has no observations or variables.
- 0 means the data set has no observations
- 1 means the data set has observations and variables.

Task: Use the **ATTRN** function to find out how many rows and columns are in a dataset.

```
3248 data _null_;
3249     dsid=OPEN('sashelp.class');
3250     if dsid ne 0 then do;
3251         totobs = ATTRN(dsid, 'NOBS'); put totobs=;
3252         totvars= ATTRN(dsid, 'NVAR'); put totvars=;
3253     end;
3254     rc = CLOSE(dsid);
3255 run;

totobs=19
totvars=5
NOTE: DATA statement used (Total process time):
```

ATTRN Function

Task: Find when a dataset was last updated. In other words, how old is the data?

```
data _null_;  
  dsid=OPEN('SASHELP.CLASS');  
  Update_Dt=attrn(dsid,"MODTE");  
  rc = close(dsid);  
  call symput('Updte', put(Update_Dt, datetime22.));  
run;  
  
title "The Class Dataset was Last Updated: &Updte";  
  
proc print data=sashelp.class(obs=7);  
run;
```

The **MODTE** argument of the **ATTRN** function makes this possible. It gets the last date the dataset was modified.

The Class Dataset was Last Updated: 25SEP08:11:34:53

Obs	Name	Sex	Age	Height	Weight
1	Alice	F	13	56.5	84.0
2	Barbara	F	13	65.3	98.0
3	Carol	F	14	62.8	102.5
4	Jane	F	12	59.8	84.5
5	Janet	F	15	62.5	112.5
6	Joyce	F	11	51.3	50.5
7	Judy	F	14	64.3	90.0



5. The PROPCASE and TRANWRD Functions

The PROPCASE function is designed to produce a character string with the proper case. Use the TRANWRD function to make the spelling of 'Drive' consistent.

patient	street_address
Ms. Jean Smith	4 Corner St.
Mr. Ronald Moore	1333 Market St.
Ms. Beth Adams	442 Glenwood Ave.
Mr. Gus Polinski	180 Cannon Dr.
Mr. Bill Pegg Jr	100 cannon Dr.

```
data p_sug.cannon_drive_2;
  set p_sug.a_patient;
  if index(upcase(street_address), 'CANNON') > 0;
  street_address = propcase(street_address);
  street_address = tranwrđ(street_address, 'Drive', 'Dr.');
```

```
run;
```

```
proc print data=p_sug.cannon_drive_2;
run;
```

patient	street_address
Mr. Gus Polinski	180 Cannon Dr.
Mr. Bill Pegg Jr	100 Cannon Dr.
Mr. Fred Gold	705 Cannon Dr.
Mr. Ed D. Cox Jr	752 Cannon Dr.
Mrs. Sue Mathis	216 Cannon Dr.

The PROPCASE function 'shifts' a character value to the proper case. The typical syntax is:

PROPCASE (argument <,delimiter(s)>)

where :

argument is a character variable or expression

delimiter specifies one or more delimiters that are enclosed in quotation marks. The default delimiters are blank, forward slash, hyphen, open parenthesis, period, and tab.

* **Tip:** If you use this argument, then the default delimiters, including the blank, are no longer in effect.

The TRANWRD function replaces or removes all occurrences of a word in a character string. The typical syntax is:

TRANWRD (source, target, replacement)

where :

source specifies the source string that you want to translate.

target specifies the string searched for in source.

replacement specifies the string that replaces target.

6. PROPCASE Function

	lastname
1	SMITH
2	MCDONALD
3	MACDOUGAL
4	O'DELL

The LASTNAMES dataset contains names that are all capitalized but cannot be properly 'fixed' using the PROPCASE function alone.

Write a program that can convert these names to a mixed case spelling.

```
data case;
  set sasuser.lastnames;
  l_name_pc=propcase(lastname);
  if substr(lastname, 1, 2)='MC' or
     substr(lastname, 1, 3)='MAC' then do;
    x = index(lastname, 'C');
    LN_1 = propcase(substr(lastname, 1, x));
    LN_2 = propcase(substr(lastname, x+1));
    Last_Name = strip(LN_1) !! strip(LN_2);
  end;
  else if substr(lastname, 2, 1)=''' then do;
    x=index(lastname, '''');
    LN_1 = propcase(substr(lastname, 1, x));
    LN_2 = propcase(substr(lastname, x+1));
    Last_Name = strip(LN_1) !! strip(LN_2);
  end;
run;
```

	lastname	l_name_pc	x	LN_1	LN_2	Last_Name
1	SMITH	Smith	.			
2	MCDONALD	Mcdonald	2	Mc	Donald	McDonald
3	MACDOUGAL	Macdougal	3	Mac	Dougal	MacDougal
4	O'DELL	O'dell	2	O'	Dell	O'Dell

7. The COMPRESS Function

The COMPRESS function returns a character string with specified characters removed from the original string.

The syntax of the **COMPRESS** function is :

compress (*source* < , *characters* > < , *modifier(s)* >)

where

- source** specifies a character constant, variable, or expression from which specified characters will be removed.
- characters** specifies a character constant, variable, or expression that initializes a list of characters.
- modifier** by default, the characters in this list are **removed** from the *source* argument. If you specify the K modifier in the third argument, then only the characters in this list are kept in the result. Specifies a character constant, variable, or expression in which each non-blank character modifies the action of the COMPRESS function. Blanks are ignored. The following characters can be used as modifiers:



The COMPRESS Function

Modifier

A or a	adds alphabetic characters to the list of characters,
C or c	adds control characters to the list of characters,
D or d	adds digits to the list of characters,
F or f	adds the underscore to the list of characters,
G or g	adds graphic characters to the list of characters,
H or h	adds a horizontal tab to the list of characters,
I or i	ignores the case of characters to be kept or removed,
K or k	keeps the characters in the list instead of removing them,
L or l	adds lowercase letters to the list of characters,
N or n	adds digits, the underscore character, and English characters to the list of characters.
O or o	processes the second and third arguments once rather than every time the COMPRESS function is called. Using the O modifier in the DATA step (excluding WHERE clauses) or in the SQL procedure, can make the COMPRESS function run much faster when you call it in a loop where the second and third arguments do not change.
P or p	adds punctuation marks to the list of characters
S or s	adds space characters (blanks, horizontal tab, vertical tab, carriage return, line feed, form feed, etc, to the list of characters,
T or t	trims trailing blanks from the first and second arguments,
U or u	adds uppercase letters to the list of characters,
W or w	adds printable characters to the list of characters,
X or x	adds hexadecimal characters to the list of characters.

The COMPRESS Function

Task: Write a DATA step to illustrate how the **COMPRESS** function can be used to look for a name that is spelled more than one way.

```
data names;
  input name $ 1-16;
  check = compress(name, '.aeiou');
  cards;
LL Beane
L.L.Bean
L.L. Bean
LL. Bena
LL Baene
;
proc print;
run;
```

Obs	name	check
1	LL Beane	LLBn
2	L.L.Bean	LLBn
3	L.L. Bean	LLBn
4	LL. Bena	LLBn
5	LL Baene	LLBn

→

The COMPRESS Function

Task: Modify the previous example by adding a third argument to the COMPRESS function. Specifically, use a 'K' to **keep** only the values in the second argument.

```
data names;  
  input name $ 1-16;  
  check = compress ( name, '. aeiou', 'K');  
datalines;  
LL Beane  
L.L.Bean  
L.L. Bean  
LL Baene  
;  
proc print;  
run;
```

Obs	name	check
1	LL Beane	eae
2	L.L.Bean	..ea
3	L.L. Bean	.. ea
4	LL Baene	aee

The COMPRESS Function

Task: Modify the third argument to the COMPRESS function so that the **case** of the second argument is **ignored**.

```
data names;  
  input name $ 1-16;  
  check = compress ( name, 'ABE', 'i' );  
datalines;  
ABCDabcd  
ABC Company  
XYZ Organization  
66 Blue Company  
;  
proc print;  
run;
```

Obs	name	check
1	ABCDabcd	CDcd
2	ABC Company	C Compny
3	XYZ Organization	XYZ Orgniztion
4	66 Blue Company	66 lu Compny

The COMPRESS Function

Task: Illustrate what happens when only one argument is used. When there is no second argument, the COMPRESS function only removes **blanks**.

```
data names;  
  input name $ 1-16;  
  check = compress ( name );  
datalines;  
LL Beane  
L.L.Bean  
L.L. Bean  
LL Baene  
;  
proc print;  
run;
```

Obs	name	check
1	LL Beane	LLBeane
2	L.L.Bean	L.L.Bean
3	L.L. Bean	L.L.Bean
4	LL Baene	LLBaene

→

21

8. ANYUPPER Function

The **North Side Clinic** has just received some new patient information. There seems to be a problem with the name field. The first and last names run together without any blanks in between them. They look like this: LindaCarter.

Write a DATA step to split names like this into 2 names.

```
data names;
  input name $ 1-16;
  x=anyupper(name, 2);
  First_Name=substr(name,1, x-1);
  Last_Name=substr(name, x);
  cards;
TimGoodberry
SmithJohn
EdCox
WilliamJohnson
;
```

```
proc print data =names;
run;
```

Obs	name	x	First_Name	Last_Name
1	TimGoodberry	4	Tim	Goodberry
2	SmithJohn	6	Smith	John
3	EdCox	3	Ed	Cox
4	WilliamJohnson	8	William	Johnson

Prior to SAS9, the problem would be solved this way...

```
data names;
  input name $ 1-16;
  short = substr(name, 2);
  cap= indexc(short, 'ABCDEFGHIJKLMNOPQRSTUVWXYZ');
  name1 = substr(name, 1, cap);
  name2 = substr(short, cap);
  cards;
TimGoodberry
SmithJohn
EdCox
WilliamJohnson
;
proc print;
run;
```

Obs	name	short	cap	name1	name2
1	TimGoodberry	imGoodberry	3	Tim	Goodberry
2	SmithJohn	mithJohn	5	Smith	John
3	EdCox	dCox	2	Ed	Cox
4	WilliamJohnson	illiamJohnson	7	William	Johnson