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_OK
  00033Y
proc print;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>Number</th>
<th>Non_0</th>
<th>New_Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0123</td>
<td>2</td>
<td>123</td>
</tr>
<tr>
<td>2</td>
<td>007_OK</td>
<td>3</td>
<td>7_OK</td>
</tr>
<tr>
<td>3</td>
<td>00033Y</td>
<td>4</td>
<td>33Y</td>
</tr>
</tbody>
</table>
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).
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.

```sas
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**.

<table>
<thead>
<tr>
<th>city_state</th>
<th>state</th>
<th>len</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>King and Queen Court House VA</td>
<td>VA</td>
<td>29</td>
<td>King and Queen Court House</td>
</tr>
<tr>
<td>Saint Mary of the Woods IN</td>
<td>IN</td>
<td>26</td>
<td>Saint Mary of the Woods</td>
</tr>
<tr>
<td>West Palm Beach FL</td>
<td>FL</td>
<td>18</td>
<td>West Palm Beach</td>
</tr>
<tr>
<td>Outer Banks NC</td>
<td>NC</td>
<td>14</td>
<td>Outer Banks</td>
</tr>
</tbody>
</table>
3. Data Cleaning with 3 Functions

<table>
<thead>
<tr>
<th>patient_id</th>
<th>name</th>
<th>county</th>
<th>state</th>
<th>zip_code</th>
<th>emergency</th>
</tr>
</thead>
<tbody>
<tr>
<td>A01101</td>
<td>Smith, Jean</td>
<td>Orange</td>
<td>NC</td>
<td>27515-2688</td>
<td>Y</td>
</tr>
<tr>
<td>A99126</td>
<td>Moore, Ronald</td>
<td>Wake</td>
<td>NC</td>
<td>27511-2414</td>
<td>N</td>
</tr>
<tr>
<td>B031073</td>
<td>Adams, Beth</td>
<td>Wake</td>
<td>NC</td>
<td>27606-4010</td>
<td>Y</td>
</tr>
<tr>
<td>B001324</td>
<td>Polinski, Gus</td>
<td>Durham</td>
<td>NC</td>
<td>27705-2102</td>
<td>N</td>
</tr>
</tbody>
</table>

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?
Data Cleaning

The SUBSTR function retrieves the first 5 digits from zip_code.

```latex
\texttt{data zip\_check (keep= name county zip5); }
\texttt{set patient;}
\texttt{length zip5 $5;}
\texttt{zip5=substr(zip\_code, 1, 5); run; proc print data = zip\_check; run;}
```

Partial PROC PRINT output. Examine ZIP5. What happened?

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

How can we fix this?

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:

\[ \text{LEFT ( argument )} \]

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:

<table>
<thead>
<tr>
<th>zip_code $ 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>27607-1234</td>
</tr>
</tbody>
</table>

Illustrate the use of the LEFT function:

\[ \text{ex . } x = \text{left (zip_code)}; \]

<table>
<thead>
<tr>
<th>zip_code $ 11</th>
<th>x $ 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>27607-1234</td>
<td>27607-1234</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>county</th>
<th>state</th>
<th>zip5</th>
<th>state_check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Fox, Mary P</td>
<td>Durham</td>
<td>NC</td>
<td>37705</td>
<td>TN</td>
</tr>
<tr>
<td>2</td>
<td>Fisher, Jo</td>
<td>New Hanover</td>
<td>NC</td>
<td>29891</td>
<td>SC</td>
</tr>
<tr>
<td>3</td>
<td>Mathis, Curt</td>
<td>New Hanover</td>
<td>NC</td>
<td>39891</td>
<td></td>
</tr>
</tbody>
</table>

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.

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

```sas
data fix_it (keep=numbers new);
  set sasuser.A_patient;
  numbers = scan(street_address, 1, ',');
  new=translate(numbers,'Oo','0','Li','1');
run;
```

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?
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, "NVARS"); 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?

```sas
* data _null_;
  dsid=OPEN('SASHELP.CLASS');
  Update Dt=attrn(dsid,"MODTE");
  rc = close(dsid);
  call symput('Update', put(Update Dt, datetime22.));
run;

title "The Class Dataset was Last Updated: &Update";
```

```sas
* 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

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

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

\[
\text{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:

\[
\text{TRANWRD ( source, target, replacement )}
\]

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.
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.
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.

```sas
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;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LL Beane</td>
<td>LLBn</td>
</tr>
<tr>
<td>2</td>
<td>L.L.Bean</td>
<td>LLBn</td>
</tr>
<tr>
<td>3</td>
<td>L.L. Bean</td>
<td>LLBn</td>
</tr>
<tr>
<td>4</td>
<td>LL. Bena</td>
<td>LLBn</td>
</tr>
<tr>
<td>5</td>
<td>LL Baene</td>
<td>LLBn</td>
</tr>
</tbody>
</table>
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.

```plaintext
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;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LL Beane</td>
<td>eae</td>
</tr>
<tr>
<td>2</td>
<td>L.L.Bean</td>
<td>..ea</td>
</tr>
<tr>
<td>3</td>
<td>L.L. Bean</td>
<td>.. ea</td>
</tr>
<tr>
<td>4</td>
<td>LL Baene</td>
<td>aee</td>
</tr>
</tbody>
</table>
The COMPRESS Function

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

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

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABCDabcd</td>
<td>CDcd</td>
</tr>
<tr>
<td>2</td>
<td>ABC Company</td>
<td>C Company</td>
</tr>
<tr>
<td>3</td>
<td>XYZ Organization</td>
<td>XYZ Orgniztion</td>
</tr>
<tr>
<td>4</td>
<td>66 Blue Company</td>
<td>66 lu Compny</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LL Beane</td>
<td>LLBeane</td>
</tr>
<tr>
<td>2</td>
<td>L.L.Bean</td>
<td>L.L.Bean</td>
</tr>
<tr>
<td>3</td>
<td>L.L. Bean</td>
<td>L.L.Bean</td>
</tr>
<tr>
<td>4</td>
<td>LL Baene</td>
<td>LLBaene</td>
</tr>
</tbody>
</table>
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;

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>x</th>
<th>First_Name</th>
<th>Last_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TimGoodberry</td>
<td>4</td>
<td>Tim</td>
<td>Goodberry</td>
</tr>
<tr>
<td>2</td>
<td>SmithJohn</td>
<td>6</td>
<td>Smith</td>
<td>John</td>
</tr>
<tr>
<td>3</td>
<td>EdCox</td>
<td>3</td>
<td>Ed</td>
<td>Cox</td>
</tr>
<tr>
<td>4</td>
<td>WilliamJohnson</td>
<td>8</td>
<td>William</td>
<td>Johnson</td>
</tr>
</tbody>
</table>

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;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>short</th>
<th>cap</th>
<th>name1</th>
<th>name2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TimGoodberry</td>
<td>imGoodberry</td>
<td>3</td>
<td>Tim</td>
<td>Goodberry</td>
</tr>
<tr>
<td>2</td>
<td>SmithJohn</td>
<td>mithJohn</td>
<td>5</td>
<td>Smith</td>
<td>John</td>
</tr>
<tr>
<td>3</td>
<td>EdCox</td>
<td>dCox</td>
<td>2</td>
<td>Ed</td>
<td>Cox</td>
</tr>
<tr>
<td>4</td>
<td>WilliamJohnson</td>
<td>illiamJohnson</td>
<td>7</td>
<td>William</td>
<td>Johnson</td>
</tr>
</tbody>
</table>