Working with Customer Contact Data

Jon Purvis, Texas Parks and Wildlife Department, Austin, TX

Abstract

Customer contact info is critical to success, but as received often contains inaccuracies and may be in different formats. Contact data also often does not have all the fields we would like to have; e.g., the county or metropolitan statistical area is not recorded. Before running any analysis or ETL processes, significant manipulation of the data may need to occur. SAS ® DATAFLUX is designed for this purpose, but many do not have access to it. In this paper I describe numerous SAS BASE data step coding techniques that I use to explore, clean, and format messy contact data. In addition, code to create some commonly missing fields is provided.

Data Structure

Documentation of the data does not always exist, or may be out of date. And just because a field was supposed to be used in a certain way, does not mean it was used as intended. Every new project should start by looking at the data structures and examining the data so that you know what you have. While you can examine the SAS tables individually by opening them and/or viewing their properties to see the structure, a report (or output file) provides a permanent reference, and is a good start on a data dictionary, if one is needed.

```
proc contents data=Work._all_
  details;

proc contents varnum data=Test;

proc datasets data=Test;
  contents varnum;
```

The first will list all files in the Work library and give details on them, while the second and third both give the details on a single file. The VARNUM option tells SAS to list the variables by their position in the file; by default, they are listed alphabetically. The only difference between using PROC CONTENTS and PROC DATASETS for this purpose is the default LIBREF in the DATA option.

Data Exploration

Getting a list of files and variables is the first step in the project. Next is exploring the data in each field. Variables are not always given the best names, and the actual use may not be the one you expect. You may expect the Gender field to hold the values “F” and “M”, but instead find that “f”, “m”, “U”, “T”, “0”, and “1” are also in it. The DriverLicenseState field may seem clear until you realize there are 487 unique values found, not the 51 you expected. And does ResidentState refer to the state they live in, or is it a Yes/No answer to whether they live in the state? Data is often messy - some would say it is always messy - and the first step to knowing what needs cleaned, and how much cleaning will be required, is to take a quick look at the data.

Proc Freq

This procedure produces frequency and cross tabulation tables, among other things. It is a quick way to get a list of all values found in a variable, and is the easiest way to find if unexpected values are present. The frequency counts are also a good tool to help decide whether a value is an uncommon answer, an outlier, or an obvious error. Proc Freq is case sensitive, so if Male has been entered as "Male", "MALE", and "MalE", it will let you know.

```
proc freq data=Test;

proc freq data=Test;
  tables BirthDate Gender;

proc freq noprint data=Test;
  tables BirthDate/out=Temp1;
  tables Gender/out=Temp2;
```

The first will produce frequency counts of all variables in the named dataset. The second produces counts for just the named variables. The third does not display the results on the screen, but instead creates a new dataset containing the frequency counts and percentages for each variable, which can then be further manipulated or exported.

Proc Means

This procedure provides descriptive statistics for numeric variables. This will let you know what to expect from the data, as well as help flag outliers. PROC FREQ will give the frequency of all values, and thus the minimum and maximum. However, PROC MEANS will give those values in a much more concise format, along with the mode, median, quantiles, mean, and confidence limits on the mean, among other things.

```
proc means data=Test;
  class Gender;
  var Age;
  output out=Temp;

proc means noprint nway data=Test;
  class Gender SchoolYear;
  var Grade;
  output out=Temp n nmiss median mean lclm uclm;
```

The first will calculate the number of observations, minimum, maximum, mean, and standard deviation of Age for all observations, as well as by Gender, and output them to the Temp dataset. The second does not show the output, but does save it to a new dataset. The output dataset only contains the statistics for combinations of Gender and SchoolYear (no separate analyses by Gender, SchoolYear, and all observations), and only the statistics listed in the output statement are kept in the output dataset.

Proc Univariate

This procedure provides a variety of descriptive and summary statistics, and as such duplicates many of the functions of PROC FREQ and PROC MEANS. However, as it only works on numeric data, it doesn't completely replace them. It can produce histogram and box plots that give a visual representation of the data, and provides measures of kurtosis, skewness, and distribution.

```
proc univariate freq normal plots data=Test;
  var Age Height Weight;
  by Gender;
  histogram Age;
```

This statement will output the basic statistics, moments, tests for location and normality, quantiles, extreme observations, and give a frequency count and percentage for all three variables broken down by Gender. It then plots a histogram of the distribution of Age, but not the other two variables.

Proc SQL

You can submit SQL (Structured Query Language) statements to SAS using PROC SQL. Depending on the task, it may be easier to use SQL, or a PROC step; much of the time personal preference and expertise will be the deciding factor.

```
proc sql;
  create table Temp1 as
    select BirthDate,
      count(IDNum) as People
    from Test
    group by BirthDate
    order by BirthDate;

  create table Temp2 as
    select Gender,
      count(IDNum) as People,
      mean(Age) as AverageAge,
      min(Age) as FirstAge,
      max(Age) as LastAge
    from Test
    group by Gender
    order by Gender;
quit;
```

The first statement calculates the frequency counts for each BirthDate, while the second calculates frequency, and the minimum, maximum, and mean age by gender. PROC SQL does not display the results to the screen, so you would need to either open the dataset or use a PROC PRINT to view them.

Data Cleansing

Once you know what you have, you can start making decisions on what are acceptable outliers, and what are errors that should be corrected. Whether the data is entered by the customer in an online application, by a clerk in a store, or by data entry staff looking at paper forms, mistakes happen. People might not understand what they are supposed to be entering, or worse, not care. Typos are a fact of life, and handwriting is often hard to read. Getting the contact info is critical, but if it is not accurate, it has no value. Here are some methods of cleaning messy data that I regularly use.

Address Standardization

Address data can be entered using abbreviations or entire words, and the abbreviations used are not always the ones accepted by the Post Office. Correcting these will ensure a better chance of contacting the person if needed, as well as enabling better record matching if that is needed. In some cases, the address can be quite large, and replacing full words with abbreviations can save space, both in the database and when printing mailing labels. The only difference between TRANWRD and TRANSTRN is that TRANSTRN allows the replacement string to have a length of zero, and thus can be used to both replace and remove characters.

```
data Test; set Test;
  Address=transtrn(Address,'STATE HIGHWAY','SH');
  Address=transtrn(Address,'COUNTY ROAD','CR');
  Address=transtrn(Address,'COUNTY RD','CR');
  Address=transtrn(Address,'PRIVATE ROAD','PR');
  Address=transtrn(Address,'FARM ROAD','FR');
  Address=transtrn(Address,'RURAL ROUTE’,'RR');
  Address=transtrn(Address,'POB ','PO BOX');
  Address=transtrn(Address,'P0B ','PO BOX');
```

Country Names

For many applications, it can be assumed that all customers are from the USA, and thus Country does not need to be stored. However, many will have customers from multiple nations, and unless they are forced to choose the Country from a list, it will be incorrect many times. Just as an example, I have seen 46 different ways of entering USA in the Country field in one database, and the variations on Canada and Mexico were much worse. The DATA statement below corrects misspellings and alternate names of countries. I have only shown that for the USA due to space constraints – contact me if you want the complete set. The first SQL statement resets the Country and State if the value of Country is a Texas county. This works well if most customers are from a single state. If customers are spread across the USA, then duplicate county names between states makes setting it to a single state problematic. The second and third SQL statements resets the Country if the value is a state name or abbreviation.

```
data Test; set Test;
  if Country in ('AMEERICA','AMERICA','AMERICIA','ESTADOS UNIDOS','PORT RICO',
    'PORTO','PORTO RICO','PUAERTO RICO','PUERTO PICO','PUERTO RICE','PUERTO RIO',
    'PUERTO ROICO','PURTO RICO','U S','U S A','U. S. .','U.S','U.S.','U.S.A',
    'U.S.A.','UMITED STATES','UNIDED STATES','UNIT STATE','UNITD STATES',
    'UNITE STATE','UNITE STATES','UNITEC STATES','UNITED STAE','UNITED STAES',
    'UNITED STATE','UNITED STATED','UNITED STATES','UNITED STATS','UNITEDSTATES',
    'UNITES STATES','UNTIED STATES','UNTIES STATES','UNTITED STATES','US',
    'US EMBASSY','US OF AMERICA','US VIRGIN ISLANDS','USSA','VIRGIN ISLAND',
    'VIRGIN ISLANDS(U.S.)'
  then Country='USA';
  if Country^='USA' and Country^='' then do;
    [snipped 300 more lines of code covering other countries]
  end;

proc sql;
  update Test
  set Country='USA', State='TX'
  where Country^='USA'
    and Country in (select distinct CountyNm as County
                    from SASHELP.ZipCode
                    where StateName='Texas');

  update Test
  set Country='USA'
  where Country^='USA'
    and Country in (select distinct StateCode as State
                    from SASHELP.ZipCode);


  update Test
  set Country='USA'
  where Country^='USA'
    and Country in (select distinct StateName as State
                    from SASHELP.ZipCode);
quit;
```

Email Addresses

People often do not want to give out email addresses because they know they will be getting marketing emails afterwards. So along with the always present typos, there will be invalid emails. The only sure way to validate an email address is to send an email to the address, but there are things that can be checked that will eliminate many bad addresses before that needs done. A rigorous test using regular expressions is substantial, and overkill for most applications. For an in-depth discussion on this topic, go to https://www.regular-expressions.info/email.html. To get a complete, but really complex regular expression (about 6,500 characters!), go to http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html.

For many of us, the following will be good enough. First, get rid of all spaces using COMPRESS. While a space can be used in an email address if between two quotes, not all systems allow it, and thus it is rare enough that it is much more likely to be invalid. Next, replace all instances of two dots with one. An email cannot start or end with a dot character, so if it does, remove it. All emails must be a minimum of six characters, so any shorter than that are invalid. Finally, test to see if it fits the rule of characters, @, characters, dot, characters. If it passes all this, chances are good that it could be an actual email address.

```
data Test; set Test;
  Email=compress(Email,' ');
  Email=tranwrd(Email,'..','.');
  if substr(Email,1,1)='.' then Email=substr(Email,2);
  if substr(Email,length(Email),1)='.' then Email=substr(Email,1,length(Email)-1);
  if length(Email)<6 then Email='';
  if prxmatch("/[^@]+@[^@]+\.[^@]+/", Email)=0 then do;
    Email='';
    BadEmail='Yes';
  end;
  else BadEmail='No';
```

Excess Spaces

Some database character fields may be set up so that values shorter than the maximum field length are padded with spaces at the end. In other cases, the user may accidentally or purposely insert extra spaces at the end of the field, or even type in one or more spaces instead of leaving it blank. In the following code, FirstName is left aligned, then all trailing blanks are removed. If the string is then blank, it returns a zero-length string. For LastName, both leading and trailing blanks are removed. They accomplish the same thing, but the STRIP function runs faster. Address has all instances of multiple blanks set to single blanks. City has all instances of two blanks replaced with a single blank. In general, COMPBL is better for this purpose than TRANWRD. As an example, if there were four blanks, the TRANWRD example would replace them with two, whereas COMPBL replaces them with a single blank. If you think there may be a variable number of blanks, always go with COMPBL. For Email, all blanks have been removed.

```
data Test; set Test;
  FirstName=trimn(left(FirstName);
  LastName=strip(LastName);
  Address=compbl(Address);
  City=transwrd(City,'  ',' ');
  Email=compress(Email,' ');
```

Incorrect Case

Changing capitalization is a common task. The caplock key was turned on accidentally, or perhaps they forgot that Yes/No answers are supposed to be entered as Y/N, not y/n. The Post Office prefers that the name and address be capitalized on mailing labels, so you change it before creating the mail merge file. You may be comparing values, so having everything in upper case means fewer comparisons. Rules for when to capitalize words differ by the style manual in use, and there's no set of rules that cover every

situation. When in doubt, the general rule recommended by The U.S. Government Printing Office Style Manual is: "*Capitalize all words in titles of publications and documents, except a, an, the, at, by, for, in, of, on, to, up, and, as, but, it, or, and nor*." In the following code, Gender is set to upper case and Status to lower case. LastName has the first letter following a space, dash, or apostrophe set to upper case – this will allow names like O'Keefe and Heany-Burns to be correctly capitalized. Country has the first letter of all words set to upper case, and then resets "And" to the proper case.

```
data Test; set Test;
  Gender=upcase(Gender);
  Status=lowcase(Status);
  LastName=propcase(LastName," -'");
  Country=propcase(Country);
  Country=tranwrd(Country,' And ',' and ');
```

Invalid Characters

A very common typo is typing in "O" or "o" when "0" was meant. Less common, but still occurring, is the opposite. While an automated fix would be difficult for addresses, it is a simple fix for Names and ZipCodes. Also seen is special characters (e.g., !@#$%^&*?-/\) in fields. While some of these may be allowable in an Address, you would never expect to see them in a State or Country field. The code below switches "0" and "O" in Name; in ZipCode it does the opposite. Since TRANWRD is case sensitive, it must do two searches to get both "O" and "o". For State, it removes all digits and the selected special characters.

```
data Test; set Test;
  Name=transwrd(Name,'0','O');
  ZipCode=tranwrd(Zipcode,'O','0');
  ZipCode=tranwrd(Zipcode,'o','0');
  State=compress(State,'1234567890!@#$%^&*?-/\');
```

Creating New Fields

Data is often collected for one reason, but other uses are quickly found. Birth date is collected, but now age is needed, as well as categorizing by adult or juvenile. You have the city of residence, but need the county or metro area. Asking customers too many questions can cause issues, and in any case, changing the database and interface may not be an option. Here is a selection of variable creation methods I find myself running on a regular basis.

Age

Birth date is handy, but most of the time we are really interested in the Age. Even then, it is often a case of whether they are adult or youth, or some age category. The following provides the Age, AgeStatus, AgeDecade, and AgeGeneration variables. Because the birth date is sometimes entered incorrectly (I have both 14 December 1776 and 8 June 3003 in one of my databases!), it is wise to check for dates that are out of range first.

```
data Test; set Test;
  if BirthDate>today or BirthDate<'01JAN1900'd then BirthDate=.;

  if BirthDate^=. then do;
    Age=floor(yrdif(BirthDate,today(),'AGE'));
    BirthYear=BirthYear;
  end;
  else do;
    Age=.;
```

```
    AgeStatus='';
    AgeDecade='';
    AgeGeneration='';
    BirthYear=.;
  end;

  if Age^=. then do;
    if 1<=Age<=17 then AgeStatus='Juvenile';
      else if Age>17 then AgeStatus='Adult';

    if 1<=Age<=17 then AgeDecade='17 or less';
      else if 18<=Age<=24 then AgeDecade='18 - 24';
      else if 25<=Age<=34 then AgeDecade='25 - 34';
      else if 35<=Age<=44 then AgeDecade='35 - 44';
      else if 45<=Age<=54 then AgeDecade='45 - 54';
      else if 55<=Age<=64 then AgeDecade='55 - 64';
      else if 65<=Age then AgeDecade='65 and over';
  end;

  if BirthYear^=. then do;
    if BirthYear<=1924 then AgeGeneration='Greatest';
      else if 1925<=BirthYear<=1945 then AgeGeneration='Silent';
      else if 1946<=BirthYear<=1964 then AgeGeneration='Baby Boomer';
      else if 1965<=BirthYear<=1976 then AgeGeneration='Generation X';
      else if 1977<=BirthYear<=1995 then AgeGeneration='Millennial';
      else if BirthYear>=1996 then AgeGeneration='Generation Z';
  end;
```

Distance

Often it is necessary to know how far someone is from a known point, such as the distance from their residence to the state park they visited. If you have the latitude and longitude of both points, it is easy to calculate the distance. If you only have the zip code for each point, then the lat/long of the zip code centroid can be gotten from the SASHelp.Zipcode dataset. As a note, PROC GEOCODE, part of SAS GRAPH, will also add the lat/long coordinates to a dataset by matching various fields, but is not covered here.

The Great Circle formula calculates the distance between two points on the surface of a sphere. The distance is calculated in miles, and requires that the lat/long be in degrees. There are other versions of this formula that use radians instead of degrees, and/or kilometers instead of miles; Google is your friend. Because the Earth is not a perfect sphere, it is only accurate to within 0.5%, which is good enough most of the time. If you think this not accurate enough, first determine the accuracy of your lat/long values, as the GPS readings may be even more inaccurate.

When better accuracy is needed, the geodetic (also known as geodesic) distance should be calculated. This is the shortest path along the ellipsoid of the Earth at sea level between one point and another SAS has created two functions that will easily calculate this for us. The GEODIST function can take degrees or radians as the input, and outputs the distance as kilometers or miles – both are set with modifiers. The ZIPCITYDISTANCE function calculates the distance in miles between the centers of two zip code centroids; the lat/long of the centroids comes from the SASHelp.Zipcode dataset.

```
data Residence; set SASHELP.ZipCode;
  keep Zip X Y;
  rename X=Long1 Y=Lat1;
data Park; set SASHELP.ZipCode;
  keep Zip X Y;
  rename X=Long2 Y=Lat2;
data Test;
  merge Test Residence Park;
  by Zip;
```

```
PI180=0.0174532925199433; /*this is pi divided by 180*/
GreatCircleDistance=7921.6623*arsin(sqrt((sin((PI180*Lat2-PI180*Lat1)/2))**2
  +cos(PI180*Lat1)*cos(PI180*Lat2)*(sin((PI180*Long2-PI180*Long1)/2))**2));
GeodeticDistance=geodist(Lat1,Long1,Lat2,Long2,'DM');
ZipDistance=zipcitydistance(HomeZip,ParkZip);
```

Full Names

Many databases use separate fields for the suffix, first, middle, and last names. For various reasons, sometimes it is needed as a single field. There are multiple concatenation functions (CAT, CATS, CATQ, CATT, and CATX), as well as the concatenation operator "||". They differ mostly in how they handle leading and trailing blanks and whether a delimiter is inserted or not. For FullName, CATX works well, but for LastNameFirst, we don't want a blank inserted between the LastName and the comma, so for it the concatenation operator (or the CATS function) works better.

```
data Test; set Test;
  if MiddleName='' then do;
    FullName=catx(' ',FirstName,LastName);
    LastNameFirst=LastName||', '||FirstName;
  end;
  else do;
    FullName=catx(' ',FirstName,MiddleName,LastName);
    LastNameFirst=LastName||', '||FirstName||' '||MiddleName;
  end;
  if Suffix^='' then do;
    FullName=catx(' ',Name,Suffix);
    LastNameFirst=LastNameFirst||' '||Suffix;
  end;
```

Metropolitan Statistical Area

Sometimes we need to group people by the location in which they live, and this is often based on the metropolitan area. The metropolitan statistical area (MSA) is a code created by the US Census Bureau for just such a purpose. There are over 300 MSAs; all non-metropolitan (rural) areas use code 0. Note that MSAs are assigned by county, not city, zip code, census block, or residence. This means that rural areas within some counties are within an MSA, while some small towns in other counties are coded rural. If you only need to know which MSA, if any, a person is near, then this is not an issue. If you are needing to designate a person as urban or rural, it may be fine, but will depend on the exact need.

The SASHelp.Zipcode dataset contains a single record for each zip code. Each zip in it is assigned to a single city, county, state, and MSA. But, because zip codes can cross city, county, and even state boundaries, there can be the occasional error using this method. Also, the SASHelp.Zipcode table only stores the MSA code – to get the MSA name you will need to look it up on the web. Finally, the SASHelp.Zipcode table in SAS EG 7.1 contains 342 MSA codes. The codes have been changed and updated slightly through time (I have seen 332 and 337 listed in places on the web), and thus it may not always match up exactly with other lists you may find. So, while there are issues, if used carefully, it can be a valuable tool.

```
data Test;
  merge Test SASHelp.ZipCode(keep=Zip MSA);
  by Zip;
  if MSA>0 then Urban='Yes';
    else of MSA=0 then Urban='No';
```

Contact Information

Your comments and questions are valued and encouraged. I can be contacted at:

Jon Purvis
Texas Parks and Wildlife Department
4200 Smith School Road
Austin, TX 78744
512-389-8193
jon.purvis@tpwd.texas.gov