SESUG 2023 Paper 160

# Generating Mock Data in SAS®

Imelda C. Go, PhD, I Go, LLC and
Abbas S. Tavakoli, DrPH, MPH, ME, University of South Carolina

## ABSTRACT

Programmers often develop SAS programming code before actual/real data are available. To fill the gap, simulated/mock data can be used to test the code and verify the code is performing as intended. Mock data can be based on real data, be generated using randomization techniques, or be a combination of both. The data will contain ideally enough test cases to test our code against. Generating realistic data with internally consistent data can be challenging. This paper will discuss considerations and include examples of a way to generate discrete mock data, while controlling the distribution of univariate and multivariate values.

## INTRODUCTION

Computer programmers need data to test their code against. If we use enough records and test cases, we can conduct performance/load/stress testing and get insights into run time, necessary disk space for output, how final reports will appear, etc. Having the data well in advance to test your programs is ideal, but this may not be possible especially when it's the first year of a project. Creating mock data by tapping into real data sets (e.g., using last year's data as the basis of this year's mock data) and simulation data are alternatives.

The following are a number of considerations when generating mock data:

- **PROBABILITIES.** In randomly generating simulated data, we may need to control how often a simulated value will appear. Controlling a univariate distribution is simpler than controlling a multivariate distribution.
  - Univariate: gender
  - Multivariate: gender and race/ethnicity
- **TEST CASES.** In the ideal, the mock data will resemble real data and have enough test cases to determine if the code is addressing all the business rules/specifications/ requirements.
- **SORT ORDER.** When generating character values, be aware of how capitalization and embedded blanks/numbers/punctuation affect the data's sort order.
  - A1, A2, A10, A20 will sort in ascending order as A1, A10, A2, A20 in SAS.
  - A01, A02, A10, A20 will sort in ascending order as A01, A02, A10, A20 in SAS.
- **REPLICABILITY OF SIMULATED DATA.** This paper will use the RANUNI random number generator for creating simulated data. The RANUNI function returns a value from a uniform distribution (0, 1). RANUNI(0) invokes the function with a seed of zero, which will cause SAS to use the time of day as the seed. You can replicate the value generated by controlling the seed (positive integer).
- **USING LAST YEAR'S DATA.** If this year's data will look just like last year's data, then that's perfect. Often, this year's processing has some changes and last year's data needs to be adjusted to become this year's mock data.
- **DATA PRIVACY.** Some of us work with data governed by privacy laws. We need to observe those laws in the process of generating/using the mock data. There is an advantage to using clearly fake values (e.g., Firstname1, Firstname2, Firstname3, …) instead of real ones (e.g., Jack, Jill).

## EXAMPLE 1: UNIVARIATE VALUES ARE EQUALLY LIKELY TO OCCUR

Let us suppose we need a variable with Y and N as possible values. If we want to assume that each value is equally likely to occur, then the probability that each response occurs is 1/2. We can do the following:

```
if ranuni(0)<=.5 then response='Y';
     else if ranuni(0)>.5 then response='N';
```

Another way of producing a selection is to create a string with the two delimited values and randomly pick one of the values with equal probability of selection.

| Code | Description |
|---|---|
| `drawstring='Y,N';` | The possible values (Y and N) are separated by a delimiter (comma in this case). The delimiter visibly sets the values apart in `drawstring` especially when possible values have variable lengths. |
| `count=count(drawstring,',')+1;` | The number of possible values is the number of commas in the `drawstring` value plus 1. `Count=2` here. |
| `randompick= ceil(ranuni(0)*count);` | The RANUNI function returns a value between 0 and 1. Because 0 and 1 are not included, the CEIL function is applied to produce a `randompick` value of 1 or 2. |
| `randomvalue=scan(strip(drawstring), randompick ,',');` | `Randomvalue` value is Y or N by randomly selecting the 1st or 2nd value delimited by the comma in `drawstring`. The selection is specified by the `randompick` value. |

## EXAMPLE 2: UNIVARIATE VALUES ARE NOT EQUALLY LIKELY TO OCCUR

Let us suppose that we want to have 3 values instead with the following probabilities of occurrence.

| Value | Probability Value Occurs | Weight (Total of weights is 10) |
|---|---|---|
| Y | .40 = 4/10 | 4 |
| N | .50 = 5/10 | 5 |
| Blank (missing) | .10 = 1/10 | 1 |

We can use the same method described above. The `drawstring` value will have 4 Y values, 5 N values, and 1 blank/missing value, which reflects the probabilities of occurrence.

```
drawstring='Y,Y,Y,Y,N,N,N,N,N,';
```

We can use the rest of the coding statements in Example 1 to complete the random selection in the `drawstring`.

```
drawstring='Y,Y,Y,Y,N,N,N,N,N,';
count=count(drawstring,',')+1;
randompick= ceil(ranuni(0)*count);
randomvalue=scan(strip(drawstring),randompick,',');
```

This technique lends itself well to generalization/automation. Let us suppose we want to generate mock data for three variables:

- GENDER variable with M (probability of 3/5) and F (probability of 2/5) values
- GENDER2 variable with equally likely X and Y values
- LEVEL variable with equally likely 1, 2, 3, 4 values (Note that with the code used, the LEVEL variable will be a character variable. You need extra coding if you want a numeric variable.)

We can use the following specifications with the above technique.

| Variable Name | Value | Probability Value Occurs | Weight |
|---|---|---|---|
| GENDER | M | 3/5 | 3 |
| GENDER | F | 2/5 | 2 |
| GENDER2 | X | .5 | 2 |
| GENDER2 | Y | .5 | 2 |
| LEVEL | 1 | .25 | 2 |
| LEVEL | 2 | .25 | 2 |
| LEVEL | 3 | .25 | 2 |
| LEVEL | 4 | .25 | 2 |

When a variable's values are equally likely, the weights for each value just need to be the same. A *weight* of two was used for all GENDER2 and all LEVEL values in this example. The *weight* value will be used with the REPEAT function below. (Syntax note: `repeat(string,n)` repeats the value of `string` *n+1* times (and not *n* times)—This is why we use `weight-1` with the REPEAT function below.)

The following is an example of how to generate a data set with 5 records that contain simulated values for each of the three variables. This example codes for a situation where one of the variables needs to be numeric instead of character.

| Code | Description |
|---|---|
| `%let numrecords=5;` | This macro variable is equal to the number of records we need (5). |
| `data attributes;`<br>`input varname $ 1-8 vartype $ 9 varlength 11;`<br>`cards;`<br>`gender  C 1`<br>`gender2 C 1`<br>`level   N 8`<br>`;` | This `attributes` data set shows that LEVEL variable should be numeric, while GENDER and GENDER2 are character variables with a length of 1.<br>Obs varname vartype varlength<br>1 gender C 1<br>2 gender2 C 1<br>3 level N 8 |
| `data chars;`<br>`retain lengthstring;`<br>`length lengthstring $1000.;`<br>`set attributes (where=(vartype='C'))`<br>`end=eof;`<br>`lengthstring=strip(strip(lengthstring)||' '||strip(varname)||' $'||strip(varlength));`<br>`if eof then do; keep lengthstring; output;`<br>`call symputx('lengthstring',lengthstring);`<br>`end;`<br>`run;` | This generates a string that we will use later to control the length of the character variables. The `lengthstring` macro variable contains the following value:<br>Obs lengthstring<br>1 gender $1 gender2 $1 |

| Code | Description |
|---|---|
| data specs;<br>input varname $ value $ weight;<br>cards;<br>gender M 3<br>gender F 2<br>gender2 X 2<br>gender2 Y 2<br>level 1 2<br>level 2 2<br>level 3 2<br>level 4 2<br>;<br><br>proc sort data=specs; by varname; | This is the specs data set that provides the information on how to create the drawstring values for GENDER, GENDER2, and LEVEL. |
| data drawstring;<br>length drawstring $200.;<br>retain drawstring '' ;<br>set specs; by varname;<br>if first.varname then drawstring='';<br>drawstring=cats(drawstring,<br>repeat(cats(',',strip(value)), weight-1));<br>if last.varname<br>then do;<br>drawstring=substr(drawstring,2,<br>length(drawstring)-1);<br>count=count(drawstring,',')+1;<br>drop value weight;<br>output; end;<br>run; | The drawstring values are created. The count value contains the total number of values that can be selected randomly.<br><br>Obs drawstring varname count<br>1 M,M,M,F,F gender 5<br>2 X,X,Y,Y gender2 4<br>3 1,1,2,2,3,3,4,4 level 8 |
| proc sql noprint;<br>select max(Length(strip(value))) INTO:<br>MaxLength from specs;<br>quit; | Macro variable MaxLength contains the length of the longest string among the values specified. This will be needed in a DATA step later. The value is 1 here. |
| proc sort data=specs; by varname;<br><br>data process;<br>length randomvalue $&MaxLength..;<br>retain seed 0;<br>set drawstring;<br>do iteration=1 to &numrecords;<br>randompick= ceil(ranuni(0)*count);<br>randomvalue=scan(strip(drawstring),randompi<br>ck,',');<br>drop value weight;<br>output; end;<br>run; | This carries out the randomization with RANUNI(0). Each time you run the statements, you will get a different data set because the seed values were set based on the time of day. If you want to create the same data set each time, then do not use 0 and control the seeds used with RANUNI. There are a number of different ways to do this.<br><br>Obs randomvalue seed drawstring varname count iteration randompick<br>1 F 0 M,M,M,F,F gender 5 1 4<br>2 Y 0 X,X,Y,Y gender2 4 1 3<br>3 3 0 1,1,2,2,3,3,4,4 level 8 1 6<br>4 F 0 M,M,M,F,F gender 5 2 5<br>5 X 0 X,X,Y,Y gender2 4 2 2<br>6 2 0 1,1,2,2,3,3,4,4 level 8 2 4<br>7 F 0 M,M,M,F,F gender 5 3 4<br>8 X 0 X,X,Y,Y gender2 4 3 2<br>9 3 0 1,1,2,2,3,3,4,4 level 8 3 6<br>10 F 0 M,M,M,F,F gender 5 4 4<br>11 Y 0 X,X,Y,Y gender2 4 4 3<br>12 2 0 1,1,2,2,3,3,4,4 level 8 4 4<br>13 M 0 M,M,M,F,F gender 5 5 1<br>14 Y 0 X,X,Y,Y gender2 4 5 3<br>15 1 0 1,1,2,2,3,3,4,4 level 8 5 1 |

| Code | Description |
|---|---|
| `proc sort data=process; by iteration;`<br><br>`proc transpose data=process out=transposed (drop=_name_);`<br>`by iteration;`<br>`id varname;`<br>`var randomvalue;`<br>`run;` | We can use PROC TRANSPOSE to get the following data set. In this data set, LEVEL is a character variable.<br><br>Obs iteration gender gender2 level<br>1 1 F Y 3<br>2 2 F X 2<br>3 3 F X 3<br>4 4 F Y 2<br>5 5 M Y 1 |
| `data ConvertToNumString;`<br>`length string $2000.;`<br>`retain string '' ;`<br>`set attributes (where=(vartype='N'))`<br>`end=eof;`<br>`string=cat(strip(string),`<br>`"num",strip(varname),"=input(",`<br>`strip(varname),",8.); rename num",`<br>`strip(varname),"=",strip(varname),`<br>`"; drop ,strip(varname),";");`<br>`if eof then do; output; call`<br>`symputx("convertToNumString",string);end;`<br>`run;`<br><br>`data process3; set process2;`<br>`&convertToNumString; run;` | Macro variable `convertToNumString` contains the following value, which are statements to convert the value to numeric.<br><br>Obs string varname vartype varlength<br>1 numlevel=input(level,8.); rename numlevel=level; drop level; level N 8<br><br>This string is used to make the conversion in the `process3` DATA step. |

Variable `level` is numeric in the final data set (`process3`) as confirmed by PROC CONTENTS.

The CONTENTS Procedure

| | | | |
|---|---|---|---|
| Data Set Name | WORK.PROCESS3 | Observations | 5 |
| Member Type | DATA | Variables | 4 |
| Engine | V9 | Indexes | 0 |
| Created | 07/05/2023 19:26:55 | Observation Length | 18 |
| Last Modified | 07/05/2023 19:26:55 | Deleted Observations | 0 |
| Protection | | Compressed | CHAR |
| Data Set Type | | Reuse Space | NO |
| Label | | Point to Observations | YES |
| Data Representation | WINDOWS_64 | Sorted | NO |
| Encoding | wlatin1 Western (Windows) | | |

Engine/Host Dependent Information

| | |
|---|---|
| Data Set Page Size | 65536 |
| Number of Data Set Pages | 2 |
| Number of Data Set Repairs | 0 |
| ExtendObsCounter | YES |
| Filename | W:\sas\sastemp_TD17544_SASOFAN01-OB-PD_\process3.sas7bdat |
| Release Created | 9.0401M7 |
| Host Created | X64_SR12R2 |
| Owner Name | COGNIA\imelda.go |
| File Size | 192KB |
| File Size (bytes) | 196608 |

Alphabetic List of Variables and Attributes

| # | Variable | Type | Len |
|---|---|---|---|
| 1 | gender | Char | 1 |
| 2 | gender2 | Char | 1 |
| 3 | iteration | Num | 8 |
| 4 | level | Num | 8 |

## EXAMPLE 3: MULTIVARIATE VALUES

We can use the univariate technique illustrated above to control the distribution of values in a multivariate situation. Here is an example of how we would write the specifications data set in the following GENDER and RESPONSE bivariate example. We list the combinations of the two variables. The combinations will be randomly selected.

| Variable Name | Value | Probability Value Occurs (Total is 1) | Weight | Description | Gender | Response |
|---|---|---|---|---|---|---|
| GENDER*RESPONSE | M+Y | .41 | 41 | Male with Y response | M | Y |
| GENDER*RESPONSE | F+Y | .32 | 32 | Female with Y response | F | Y |
| GENDER*RESPONSE | M+N | .17 | 17 | Male with N response | M | N |
| GENDER*RESPONSE | F+N | .10 | 10 | Female with N response | F | N |

In the specifications data set, we extended the variable name to include the two variables GENDER and RESPONSE delimited by *. The value column contains the values for the two variables delimited by +. Once we reworked the specs this way, we can use these specs with the same code we used in the univariate situation.

## CONCLUSION

The SAS programming language offers different tools to help us with our work. We can develop solutions that help us achieve the desired result faster.

## CONTACT INFORMATION

Imelda C. Go
I GO, LLC
igoforwork@gmail.com

Abbas S. Tavakoli
University of South Carolina
abbas.tavakoli@sc.edu

## TRADEMARK NOTICE