What are some examples of user-defined formats, also known as value labels, in SAS tutorials?

Authored by

stats writer

June 24, 2024

User-defined formats, also known as value labels, in SAS tutorials refer to a feature in the SAS programming language that allows users to create their own custom formats for data values. These formats are essentially a set of instructions that specify how data values should be displayed or categorized in a dataset. Examples of user-defined formats in SAS tutorials include converting numeric values to categorical labels, assigning specific colors or symbols to data values, and creating custom date or time formats. These formats can be used to enhance the readability and interpretation of data in SAS programs and analyses.

Creating New Formats with PROC FORMAT

Recall from the Informats and Formats tutorial that a *format* in SAS controls how the values of a variable should "look" when printed or displayed. For example, if you have a numeric variable containing yearly income, you could use formats so that the values of those variables are displayed using a dollar sign (without actually modifying the data itself).

There will be some situations where SAS built-in formats do not fit your needs - for example, nominal and ordinal categorical variables. In this case, you can create your own formats. This is done using the `PROC FORMAT` procedure.

The general form of `PROC FORMAT` is:

```
PROC FORMAT;
VALUE format-name
Data-value-1 = 'Label 1'
Data-value-2 = 'Label 2';
VALUE format-name-2
Data-value-3 = 'Label 3'
Data-value-4 = 'Label 4';
.....;
RUN;
```

The first line is the start of the proc step. The procedure we want to execute is `PROC FORMAT`. The next line starts with a `VALUE` keyword, followed by the name of the format you want to create. You can name the format whatever makes sense to you, but it must:

start with a letternot end in a number (0-9)have a maximum length of 32 charactersNOT match the name of any variables in your data set

Note that there is no semi-colon after the format name. The next set of lines assigns labels to the values of the variable in your dataset. You can create as many labels as you want and when you are finished don't forget the semi-colon after the last label. End the `PROC FORMAT` with a `RUN`

statement and a semi-colon.

Typically, you will assign a unique value label to each unique data value, but it's also possible to assign the same label to a range of data values.

Creating labels for each data value

The most common way of labeling data is to simply assign each unique code its own label. Here, the format `LIKERT_SEVEN` assigns distinct labels to the values 1, 2, 3, 4, 5, 6, 7.

```
PROC FORMAT;
VALUE LIKERT_SEVEN
1 = "Strongly Disagree"
2 = "Disagree"
3 = "Slightly Disagree"
4 = "Neither Agree nor Disagree"
5 = "Slightly Agree"
6 = "Agree"
7 = "Strongly Agree";
RUN;
```

Creating labels that apply to more than one data value

We may want to use the same value for more than one numeric code. We can do this by listing all of the values (separated by commas) to assign a given label. Format `LIKERT7_A` assigns the label "Disagree" to values 1, 2, 3; and assigns the label "Agree" to values 5, 6, 7.

```
PROC FORMAT;
VALUE LIKERT7_A
1,2,3 = "Disagree"
4 = "Neither Agree nor Disagree"
5,6,7 = "Agree";
RUN;
```

Setting the same label to a range of data values

If a numeric variable represents ordinal codes (or has some discernable order), you can assign the same label to multiple codes in a range. Format `LIKERT7_B` assigns the label "Disagree" to values 1 through 3, and assigns the label "Agree" to values 5 through 7.

```
PROC FORMAT;
```

```
VALUE LIKERT7_B
1-3 = "Disagree"
4 = "Neither Agree nor Disagree"
5-7 = "Agree";
RUN;
```

You can also use the keywords LOW and HIGH when assigning labels to a variable with continuous values. Here, format `INCOME`:

applies the label "Low" to all data values less than 20,000;applies the value "Middle" to all data values greater than or equal to 20,000 and less than 60,000;and applies the value High" to all data values greater than or equal to 60,000.

```
PROC FORMAT;
VALUE INCOME
LOW -< 20000 = "Low"
20000 -< 60000 = "Middle"
60000 - HIGH = High";
RUN;
```

Using the word OTHER to specify all other data values

```
PROC FORMAT;
VALUE RACE
1 = "White"
2 = "Black"
OTHER = "Other";
RUN;
```

Creating labels for character variables

Value labels can also be applied to character/string data values. The most important differences are:

The name of the format must start with a dollar sign ($)The code values (on the left of the equals signs) must be quoted.

```
PROC FORMAT;
VALUE $GENDERLABEL
"M" = "Male"
"F" = "Female";
```

```
RUN;
```

After the formats have been created using PROC FORMAT, they must still be applied to the data. This can either be done temporarily, by adding the labels during a PROC step, or be done permanently, by applying the labels in a data step.

Example: Permanently assigning labels to coded categorical variables

Our sample dataset has several categorical variables that, without formats, are hard to look at and know what the value represents. These include the variables Gender (which has values 0=male, 1=female), Athlete (which has values 0=non-athlete, 1=athlete), and Smoking (which has values 0=nonsmoker, 1=past smoker, 2=current smoker). Let's create formats for each of these sets of labels.

```
PROC FORMAT;
VALUE GENDERCODE
0 = 'Male'
1 = 'Female';
VALUE ATHLETECODE
0 = 'Non-athlete'
1 = 'Athlete';
VALUE SMOKINGCODE
0 = 'Nonsmoker'
1 = 'Past smoker'
2 = 'Current smoker';
RUN;
```

Again, note the placement of a semi-colons:

After `PROC FORMAT`After assignment of the last label for each setAfter the `RUN` statement.

Once you've created the format, you still have to assign it to your variable. Let's assign the new formats to their respective variables, so that when we look at the data or output, we see the labels instead of the codes.

```
DATA sample_formatted2;
SET sample;
FORMAT gender GENDERCODE. athlete ATHLETECODE. smoking SMOKINGCODE.;
RUN;
```

The syntax above creates a new dataset called `sample_formatted2` that is a copy of `sample`. We used the `FORMAT` statement to assign the previously created format called `GENDERCODE.` to the variable gender. Note that the format name ends with a period. When you add the period after the format name, that is an indication to SAS that `GENDERCODE.` is a format and not a variable.

Storing Formats to a Library

If you assign a format to a variable in a data step, the format will stay with the variable for the rest of the SAS session. However, SAS will not permanently store the definitions for user-defined formats between sessions. For example, SAS will know that the variable gender in our sample dataset is assigned to the format `GENDERCODE.`, but it won't know what the `GENDERCODE.` format is until you tell it. If you try to use your dataset with user-defined permanent formats, SAS won't be able to execute any statements on the dataset until you define your user-defined formats.

There are four ways to deal with this:

Option 1: Manually execute PROC FORMAT at the start of each SAS session

Each time you launch SAS, manually run your PROC FORMAT code before running any data steps or proc steps that reference your user-defined formats.

This approach is simple, but can be tedious if you have many user-defined formats, or want to reuse format definitions between projects.

Option 2: Permanently store your format definitions in a SAS library

Just as SAS datasets can be permanently saved in a SAS library and re-used later, you can permanently save user-defined formats in a SAS library for later reuse. If you find yourself working in a dataset with many user-defined variables, or if you want to reuse your user-defined formats on different datasets, this option will be the easiest for you.

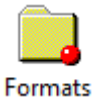
The first step is to save the format definitions to your SAS library. In the PROC FORMAT statement, the LIBRARY keyword lets you specify the name of a library in which to save the definitions:

```
LIBNAME tutorial "C:/Documents/tutorial";

PROC FORMAT LIBRARY=tutorial;
VALUE GENDER
1 = "Male"
2 = "Female";
```

```
VALUE YN
1 = "Yes"
2 = "No";
RUN;
```

In this program, we first define a SAS library called *tutorial* that's mapped to the file folder `C:/Documents/tutorial`. This is where our formats will ultimately be saved. The addition of the `LIBRARY` keyword to the PROC FORMAT statement tells SAS to store the formats to the library called *tutorial.* After running the above code, you can check to make sure that your formats have been saved by looking in the Explorer window and navigating to the library where you saved your formats. You should see a new icon in that library's contents:



This will save the formats to your library. However, in order to actually make use of these formats during a SAS session, there's an extra step you'll need to do at the start of each SAS session.

By default, SAS will only look in the WORK library (i.e., the working memory of the current SAS session) for formats, regardless of whether or not you've executed your LIBNAME statement that points to the library containing your formats. In order to access the format definitions stored in other SAS libraries, you must run `OPTIONS FMTSEARCH` at the start of your SAS session:

```
LIBNAME tutorial "C:/Documents/tutorial";
OPTIONS FMTSEARCH=(tutorial);
```

The `FMTSEARCH` system option tells SAS where to look for format definitions. Note that if you have format definitions stored multiple SAS libraries, you can specify the names of multiple SAS libraries in the parentheses after `FMTSEARCH` (separate the names by spaces). SAS will look for format definitions in the libraries you specify, and it will search the libraries in the order you list them. For example:

```
LIBNAME tutorial "C:/Documents/tutorial";
LIBNAME school "C:/Documents/school";
OPTIONS FMTSEARCH=(tutorial school work);
```

In this case, SAS will first look for format definitions in the library called *tutorial*. If it can't find a matching definition in the library tutorial, it will then look in the library called *school*; and if it can't

find a matching format in the library called school, it will look in the WORK(i.e., temporary) library.

Option 3: Use a global statement to load format definitions stored in another SAS script

As an alternative to saving formats in a SAS library, you can save your PROC FORMAT code in its own SAS script file, separate from the SAS script(s) where you will do your "main" analysis. Then at the start of each SAS session, you will load those definitions by using an `%INCLUDE` statement to execute the format definition script.

Let's suppose that we have two SAS script files:

my_format_definitions.sas - a file containing our PROC FORMAT codemy_data analysis.sas - a file where we want to use our formats in a data step

my_format_definitions.sas:

Suppose this file is saved in the folder `C:/Documents/tutorial`.

```
PROC FORMAT;
VALUE agreement
1 = "Disagree"
2 = "Neither agree nor disagree"
3 = "Agree";
RUN;
```

my_data_analysis.sas:

```
%INCLUDE 'C:/Documents/tutorial/my_format_definitions.sas';
```

DATA test;
INPUT id $ v1;
FORMAT v1 agreement.;
....
RUN;

Here the SAS keyword `%INCLUDE` tells SAS to execute the code found in the file my_format_definitions.sas. It behaves exactly as if you had manually executed the PROC FORMAT code. If the execution was successful, then the Log window should say something like

```
NOTE: Format AGREEMENT has been output.
```