Paper SM03

Elevating SAS Macros and Automation: Leveraging SAS Studio Custom Steps & Flows

Sudeshna Guhaneogi and Anand Phand, SAS R&D, Pune, India

ABSTRACT

Many organizations have extensive SAS® macro libraries, yet their use remains confined to traditional programming environments. This paper shows how SAS Custom Steps and Flows can unlock these assets by converting existing macros into reusable, parameter-driven components.

The approach enables non-programmers to run complex analyses with minimal training, improving efficiency, collaboration, and access to advanced analytics. Practical examples demonstrate how this integration transforms SAS macros into intuitive, user-friendly tools that support scalable and inclusive data-driven decision-making

CUSTOM STEPS

Custom Steps enable users to execute SAS code through a graphical user interface (GUI), allowing parameters to be provided without directly interacting with the underlying program. SAS programmers can share these steps across the organization, empowering users to run standardized, self-service routines. Custom Steps are available exclusively in SAS Viya® versions of SAS Studio.

SAS Viya comes with dozens of built-in custom steps, and dozens more are freely available on the SAS Studio Custom Steps github page: https://github.com/sassoftware/sas-studio-custom-steps.

A key use case is to repurpose an existing SAS macro to generate outputs dynamically based on user-provided inputs.

BEGIN WITH A MACRO PROGRAM

We begin with a simple program that produces a Kaplan–Meier survival plot by a group. When executed as a standalone program, it is shown below (Program 1):

```
ods noproctitle;
ods graphics / imagemap=on;
%LET Dataset = SASHELP.BMT;
%LET Time = T;
%LET Status = Status;
%LET CenVal = 0;
%LET Strata = Group;

/* Run the Lifetest to get the output */ ODS select SurvivalPlot;
proc lifetest data = &Dataset plots = (survival);   time &Time *
&Status(&CenVal);   strata &strata; run;
```

Program 1: Producing a Kaplan-Meier Plot based on User Input

We want to publish this out as a tool for users to view a K-M curve with their own data and selections, so we will go design a custom step.

DEVELOPING THE CUSTOM STEP

We'll start by using the custom step quick chart (Figure 1, green selections).

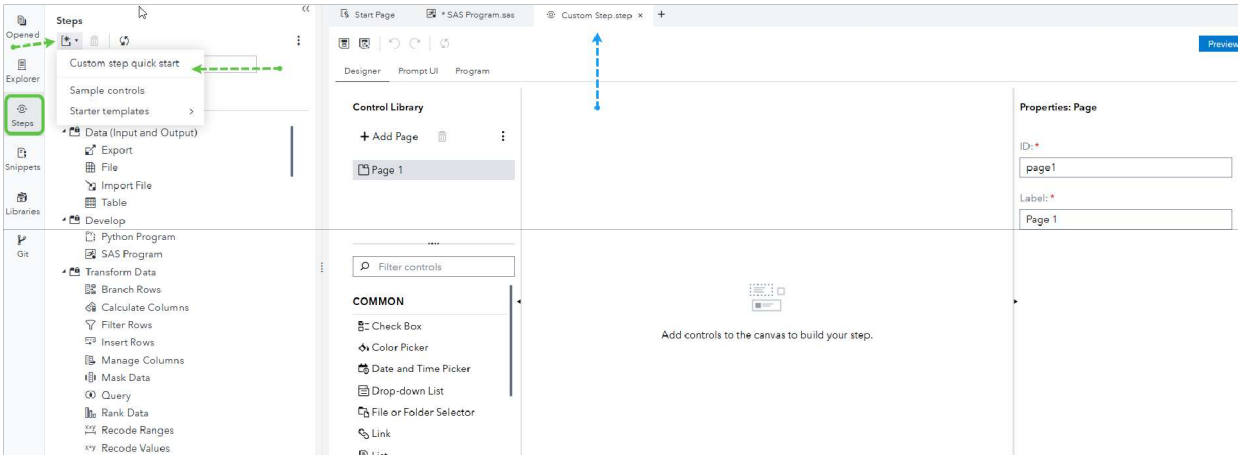

Figure 1: Custom Step QuickStart

This palette enables us to build and configure our steps. The first tab, Designer, is used to define the graphical user interface (GUI). The second tab, Prompt UI, contains the underlying UI definition written in JSON; this tab is auto-generated and can typically be ignored. The third tab, Program, is where we paste the code from Program 1, remove the existing macro variable assignments, and replace them with values captured through the GUI (Figure 2).

As a best practice, new macro variables can be created to capture inputs from the GUI and then pass them to the existing macro variables. This approach clearly distinguishes GUI-driven inputs, although direct assignment is also possible if preferred.

Figure 2: SAS Code with Macro Assignments

We are now ready to begin working on the Designer page. In this step, we need to capture five values through the GUI and pass them to the program. As a first step, we will update the page label to a more descriptive name and add the Input Table selector (Figure 3)

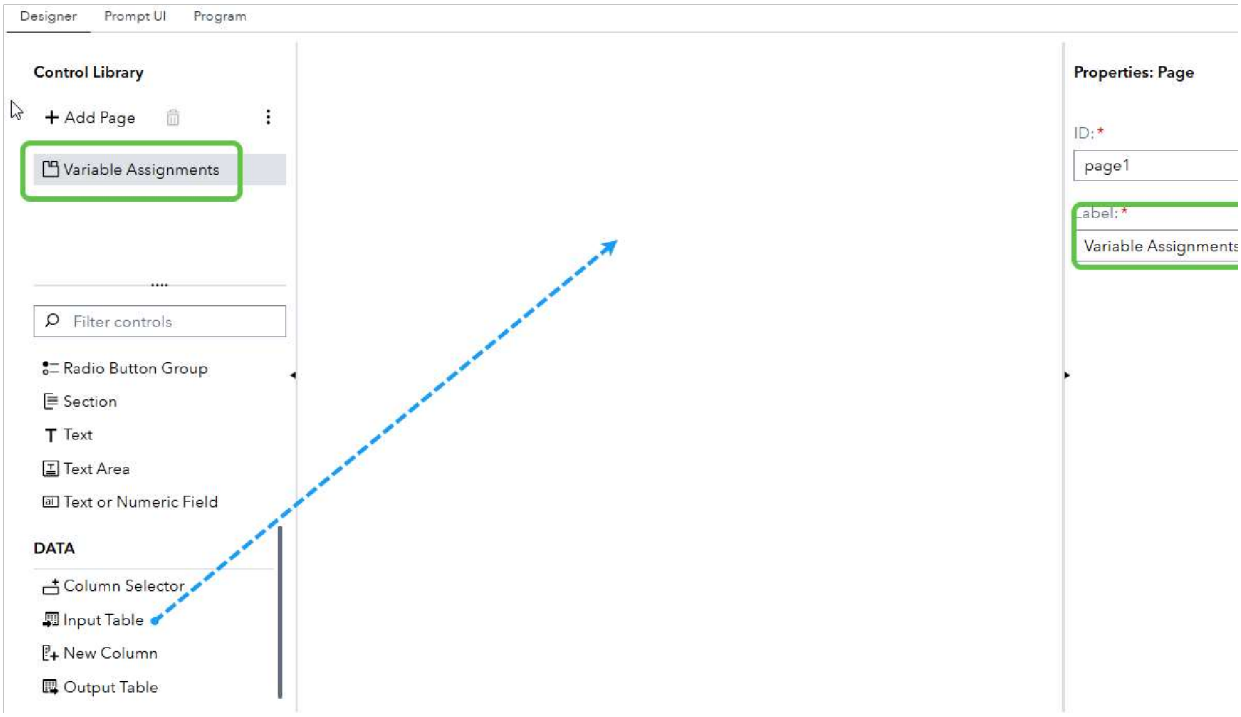

Figure 3: Add an Input Table

We need to assign some values after the table is added (Figure 4).

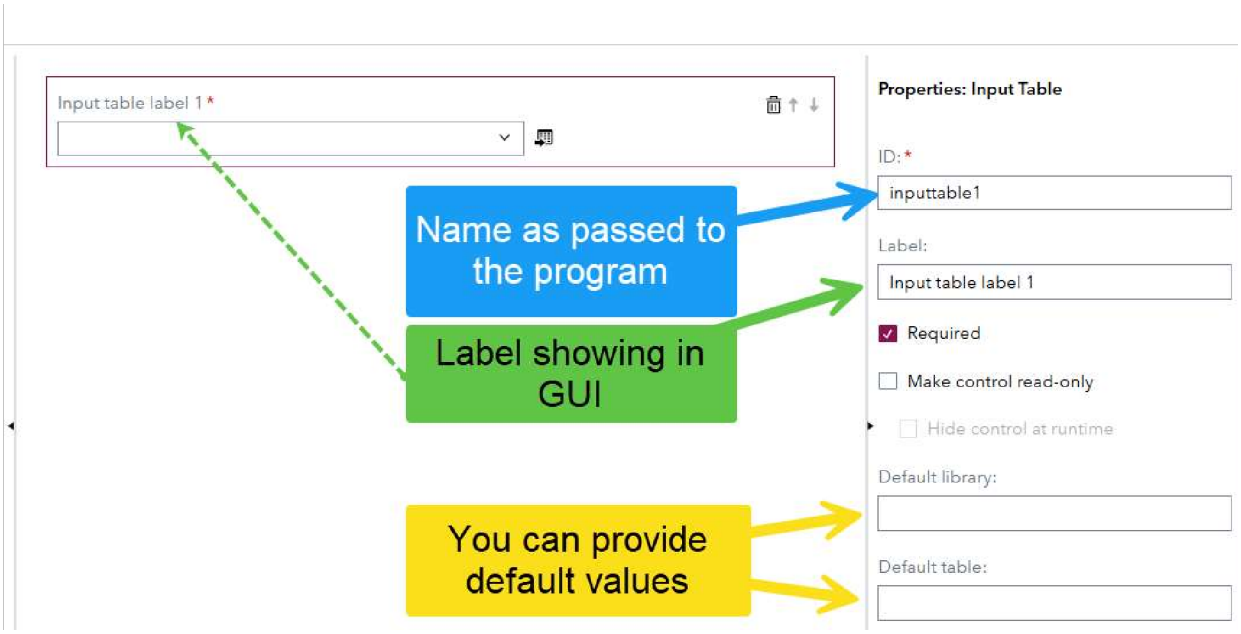

Figure 4: Input Table Properties

The ID specifies the name of the variable passed from the GUI to the program. As shown in Figure 2, this is defined as a _Dataset, which we will use here. The Label determines what is displayed in the GUI, so we will name it Data Table. While it is possible to define a default library and table, we will not do so in this example. Since this field is required, we will keep the option selected, and the red asterisk will remain next to the label.

Next, we add a Column Selector for the time variable, as shown in Figure 5.

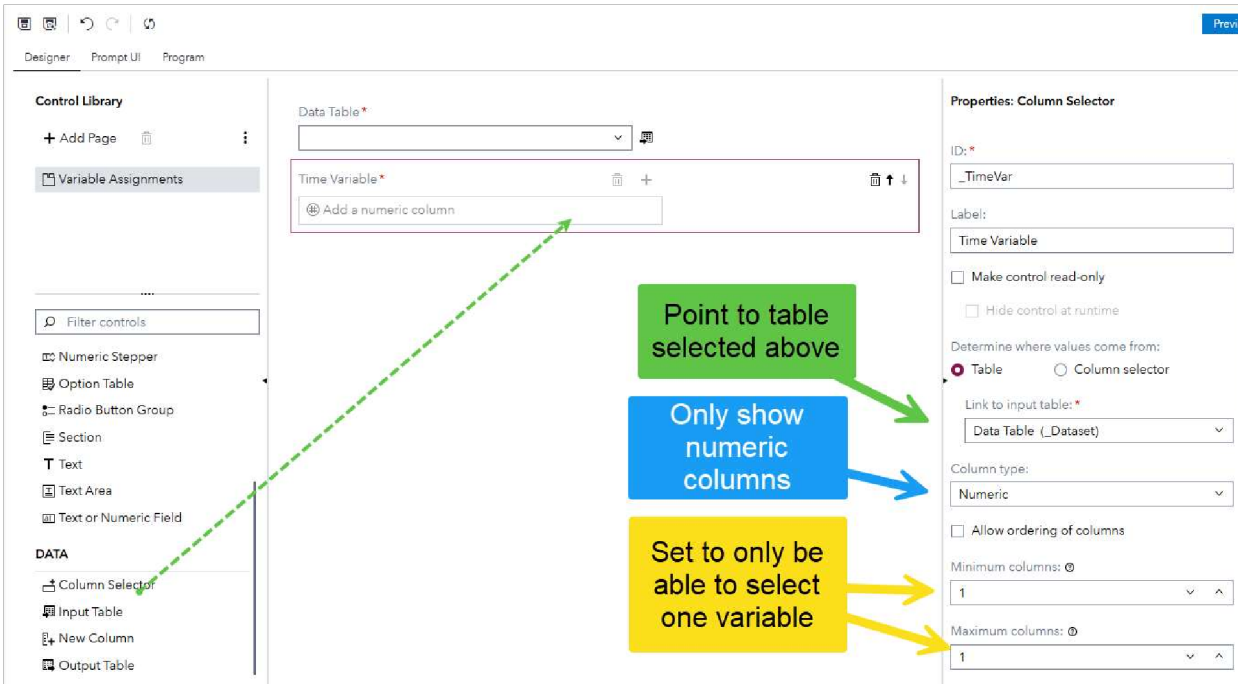

Figure 5: Column Selector

As with the Table Selector, the Column Selector requires both an ID and a Label. We will set these to _TimeVar and Time Variable, respectively. This column will be selected from the previously specified input table, and only numeric fields will be displayed. Since only a single column is required, we will set both the minimum and maximum selections to one. Remember to save your work as you proceed.

We will repeat this process for the status variable, assigning the ID _StatusVar and labeling it Status Variable. This will also be a numeric selection with exactly one column chosen. While the macro could be designed to handle either numeric or character variables using conditional logic (for example, %IF statements), we will keep this example simple and restrict the input to numeric values.

Next, we identify the value used for censoring within the status variable. To do this, we add a Drop-down List control and configure it as a dynamic list (Figure 6). This field will be marked as required, and the Status Variable will be selected as the source column. We will assign the ID _CenVal. Although a predefined list of values could be provided, using a dynamic list offers greater flexibility.

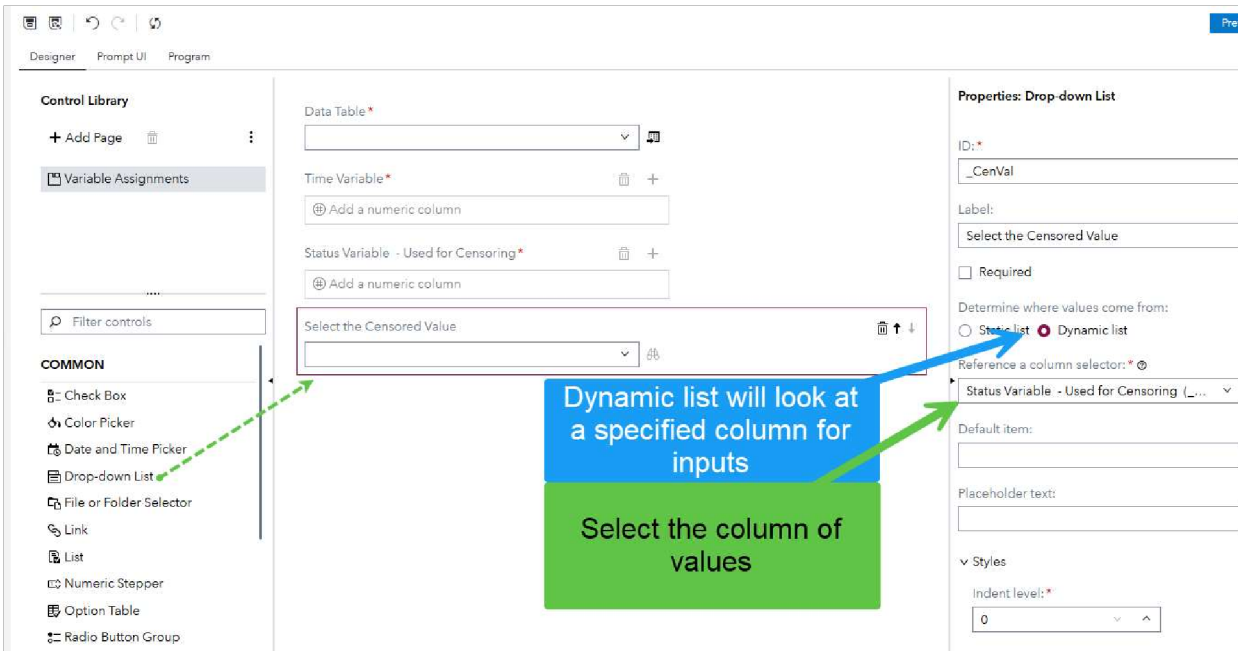

Figure 6: Drop-down List

The final step is to capture the stratification variable by adding another Column Selector with the ID _GroupVar. For this selector, we will allow all column types. With this, the GUI design is complete, and we can preview the layout by clicking the blue Preview button in the upper-right corner of the palette.

EXECUTING THE STEP

At this stage, we are ready to open and test the step. The step should appear under the Shared tab in the Steps menu; you can quickly locate it using the filter option. Once found, right-click on the step name and select Open in Tab to execute it (Figure 7).

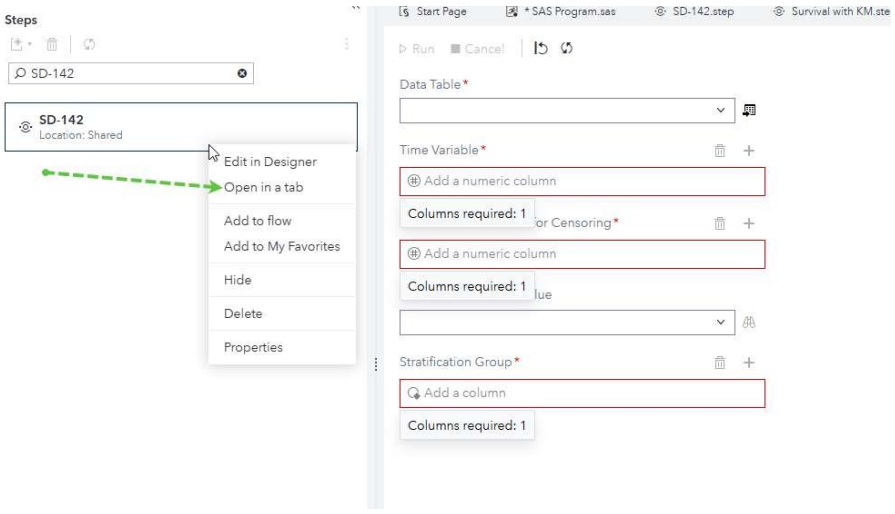

Figure 7: Opening a Step

We will fill out the step using the SASHELP.BMT dataset by following our prompts. Using the obvious selections, we get the inputs in Figure 8.

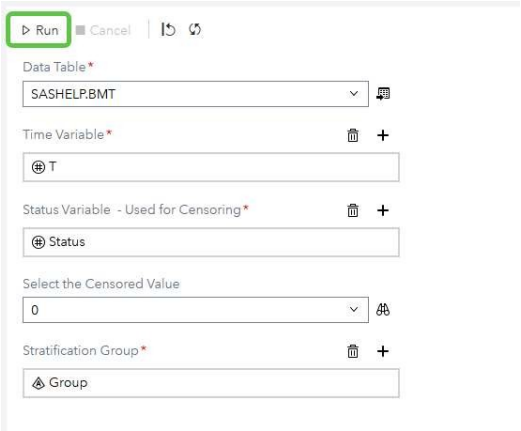

Figure 8: Entering Inputs

We can now execute the program and review the results. Some debugging may be required; a common issue is incorrectly assigning IDs that do not align with the macro variables in the program. If everything is configured correctly, the output should resemble that shown in Figure 9.

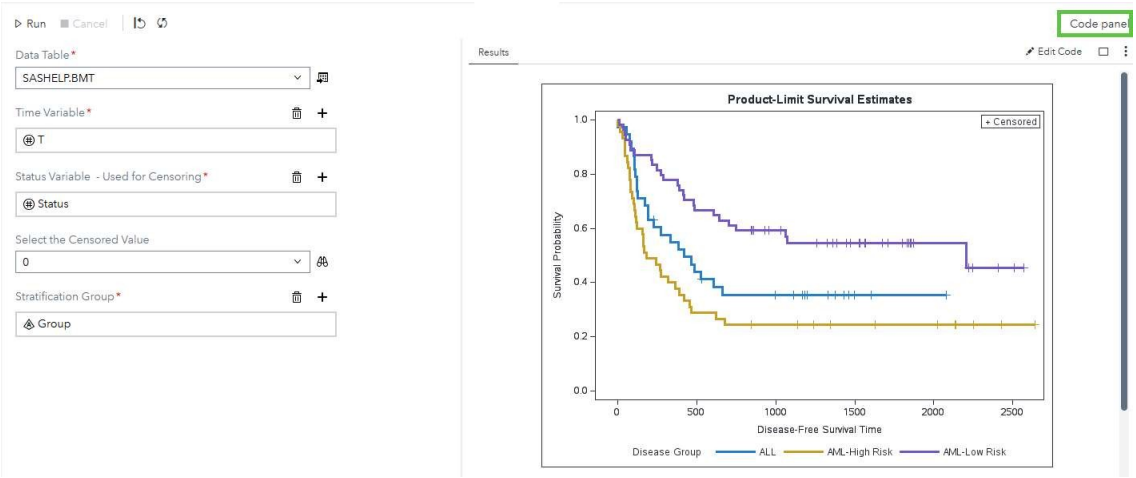

Figure 9: Final Output

By default, the results window opens automatically after execution. To review the underlying processing, you can click the Code Panel button (Figure 9, highlighted in green), which opens the Results, Log, and Code tabs for further inspection. Note that a substantial amount of wrapper code is generated automatically; in this example, 17 lines of code in the Program tab resulted in 294 lines of executed code.

Custom Steps provide a powerful way to share macro-based code with users who prefer not to modify the underlying logic but still require control over input parameters. In addition, Custom Steps—along with existing SAS steps—can be combined and orchestrated within a Flow to create end-to-end executable workflows.

COMBINING STEPS IN FLOWS

Whether you develop your own Custom Steps or leverage those provided by SAS, you can design process flows that combine multiple steps into repeatable workflows that users can execute without ever interacting with the underlying code. Figure 10 illustrates this approach, showing a flow built to generate a plot of the number of adverse events per patient, grouped by treatment.

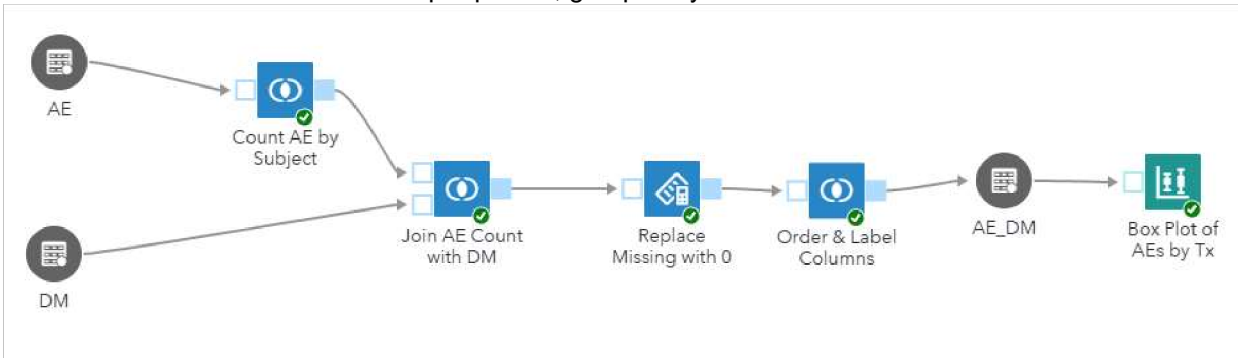

Figure 10: Example Flow

The first thing we did was add the data table (Figure 11) from the SAS steps tab and specify the STDM.AE table. In this flow, we also added the DM table, since subjects without AEs don't show up in this table.

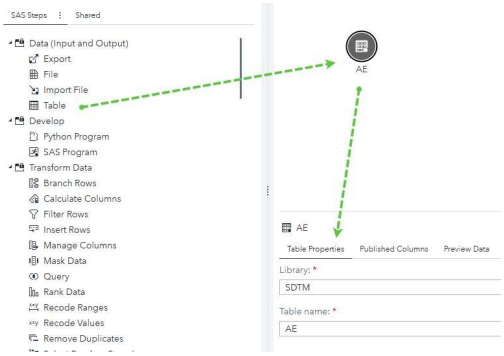

Figure 11: Add a Table

Before joining the tables, we just want to get the count of the number of records per subject in the AE table. As with anything analytical, there are multiple ways to accomplish this – we went with the SQL query route (Figure 12) by adding a column called AE_Count that was defined as count(*) and grouped by USUBJID. This will create a temporary output table that we will not write out with a name. Clicking on the Node tab of this query allows us to rename the box in the flow to make it more readable.

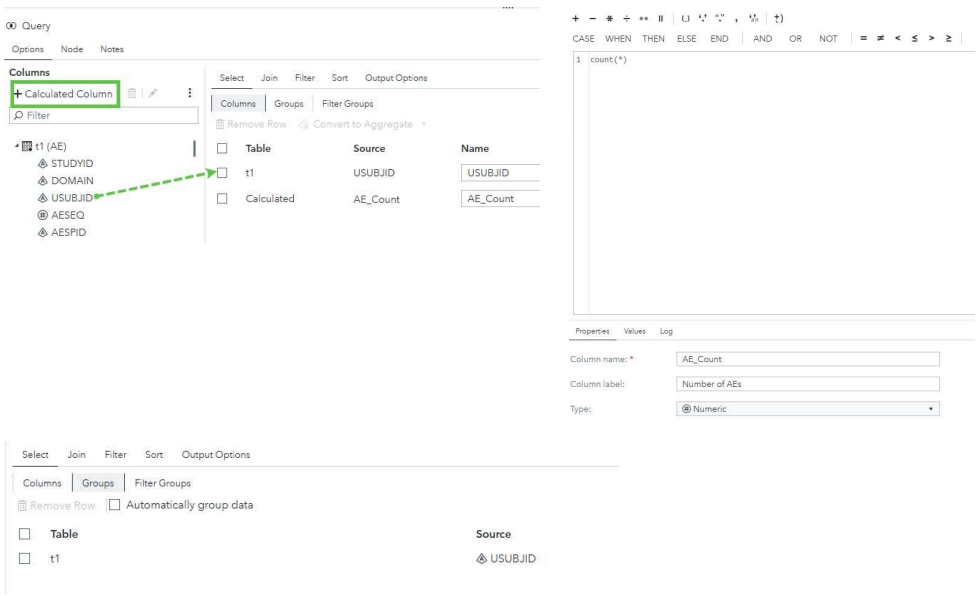

Figure 12: Create a calculated column using the Query step.

Now, we can merge in the demographics table. We used the Query step again, to avoid having to sort before doing a merge (Figure 13). When you drop in the Query box, you need to right-click on it and "Add input port" to join two datasets together. Also in this join, we selected only the variables from the DM table that we wanted in the final analytics table. The key thing to do is to make sure you are joining the AE records to the entire DM table. Also, be sure to be selecting the USUBJID from the DM table (t2) Since the AE table was joined to the top port, it is table 1, so in this instance we do a Right Join by clicking on the overlapping circles and selecting Right Join.

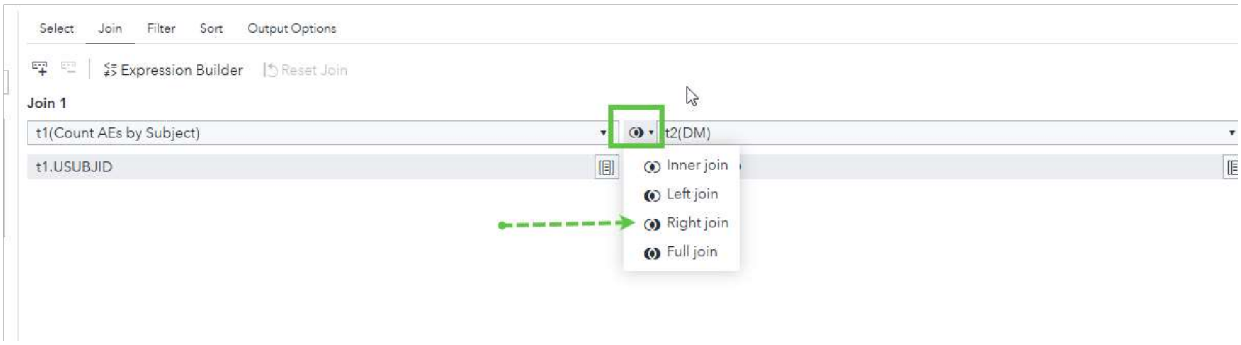

Figure 13: Configuring the Join

Run the flow at this point, and right click on the output port of the query we just made. You will see that the subjects with no adverse events will show up with a missing value for AE_Count. We could have fixed that in the query with a calculated column using the coalesce function, but here we are trying to make it easier to follow what is happening with smaller steps. In this case, we will use a Calculated Column step (Figure 14) to make the AE_Count column the max of the existing value and 0.

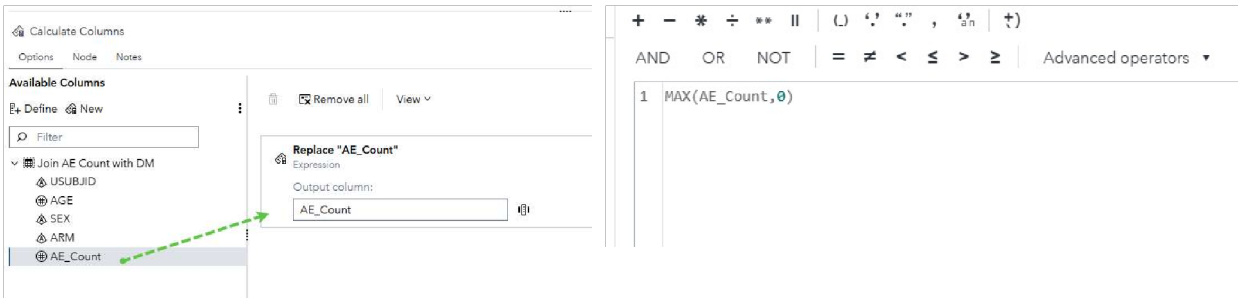

Figure 14: Calculated Column

Finally, we will add in one last Query to allow us to edit the order of the variables, drop out any we don't want, and update the labels (Figure 15). We will write that table out to the Work library by connecting a Table to the output port.

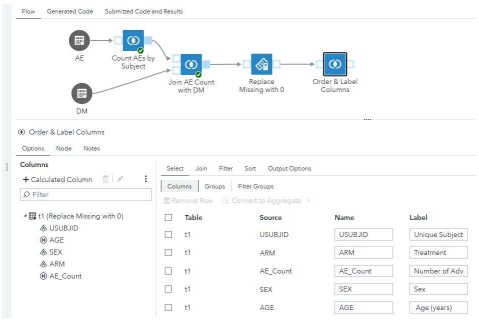

Figure 15: Final column cleanup

The last step is to connect to the visualization of choice, which is pretty self-explanatory. At this point, we have the flow in figure 10, and have created a reusable process flow that non-programmers can execute as need be.

CONCLUSION

Custom Steps provide an effective way to transform existing macros into standalone, reusable components that non-programmers can easily interact with and that can be deployed across the organization. These steps can also be combined with SAS-provided steps—and those shared by the wider programming community—to create more complex and scalable flows. Overall, Custom Steps enable organizations to fully leverage the value of their existing macro libraries and coding assets.

RECOMMENDED READING

SAS Studio Custom Steps Github: https://github.com/sassoftware/sas-studio-custom-steps

Custom Steps QuickStart Tutorial: https://www.youtube.com/watch?v=yKdCwNLEhUI

A workshop on Flows: https://github.com/Pdsaslife/PHUSE_HOW_2024

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Sudeshna Guhaneogi
SAS R&D
Sudeshna.guhaneogi@sas.com

Anand Phand
SAS R&D
anand.phand@sas.com