PharmaSUG 2025 - Paper SD-142

Share your Macros and Programs with SAS Studio Steps and Flows

Jim Box and Pritesh Desai, SAS Institute

ABSTRACT

You've probably got a huge macro and programming library at your disposal, but no way of leveraging all of that capital in any way but inside a SAS® program. We'll look at how using SAS Custom Steps and Flows will allow to unlock the full potential of your work. By leveraging these capabilities, non-programmers can seamlessly run analyses and other complex processes that you built with minimal training and intervention on your part. This integration not only enhances collaboration but also democratizes access to powerful analysis tools, enabling a broader audience to contribute to data-driven decision-making. Through practical examples and detailed guidance, this paper showcases the simplicity and efficiency of converting SAS macros into user-friendly tools, ultimately fostering an inclusive and efficient research environment.

CUSTOM STEPS

Custom steps are a tool to allow users to execute SAS code by using a Graphical User Interface (GUI) to provide parameters without and execute a program without ever having to interact with the code. SAS Programmers can share these across the organization, allowing users to run self-service routines. Custom steps are a feature available only in the 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.

One particular use case of interest would be to leverage an existing SAS macro program and create some output based on a user's input.

START WITH A MACRO PROGRAM

As a starting point, we'll look at a simple program that makes a Kaplan-Meier survival plot with groups. As a standalone program, it would look like this (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).

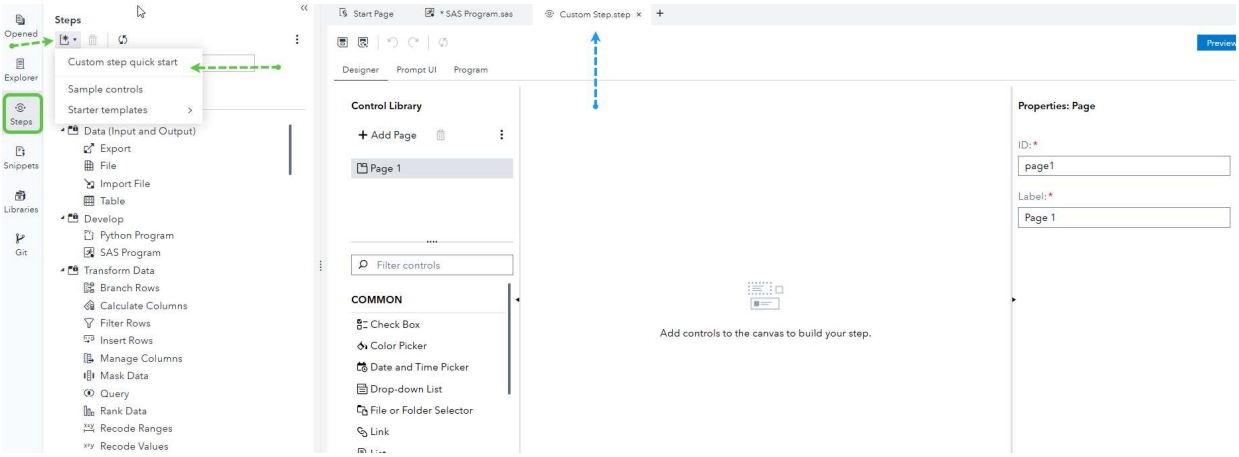

Figure 1: Custom Step QuickStart

This palate allows us to build out our steps. The first tab is the Designer, where we will set up the GUI. The second tab is the Prompt UI, which is where the UI is actually written. This will be in JSON, and you can ignore this tab, as it is automatically generated. The third tab is the Program, which is where we will paste the code from Program 1, remove the existing macro variable assignments, and replace them with ones we will capture with the GUI (Figure 2). As a best practice, I like to make new macro variables that will be captured by the GUI and passed to the existing macro variables, to clarify what comes from the GUI, but you could directly assign them if you want.

Figure 2: SAS Code with Macro Assignments

Now we are ready to start working in the designer page. We have 5 values to capture in the GUI and assign to the Program. First up, we will update the page label to a more useful value and bring in the Input Table selector (Figure 3).

Figure 3: Add an Input Table

Once the table is added, we need to assign some values (Figure 4).

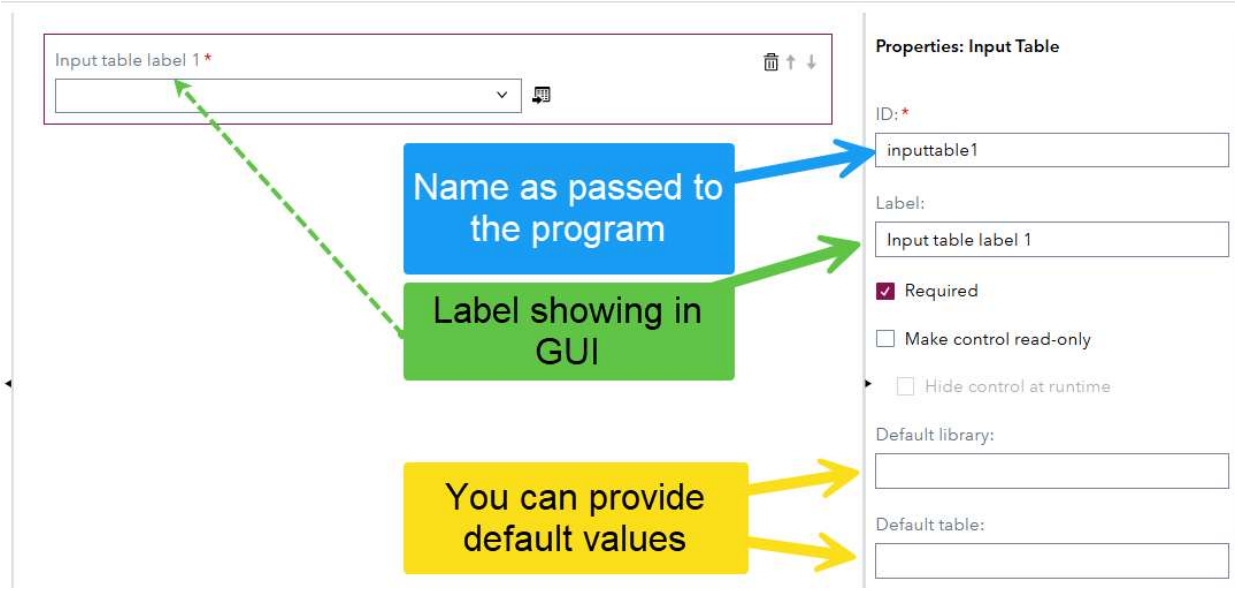

Figure 4: Input Table Properties

The ID will be the name of the variable that the GUI passes to the program. In Figure 2, we defined that as **_Dataset**, so we will enter that. The Label is what shows up in the GUI, so we will just call it **Data Table**. Note that we can set up a default library and also a default table, but we won't do that here. This is a required item, so we will keep the box checked and the red asterix will remain next to the label.

Next, we will add a Column Selector for the Time variable (Figure 5).

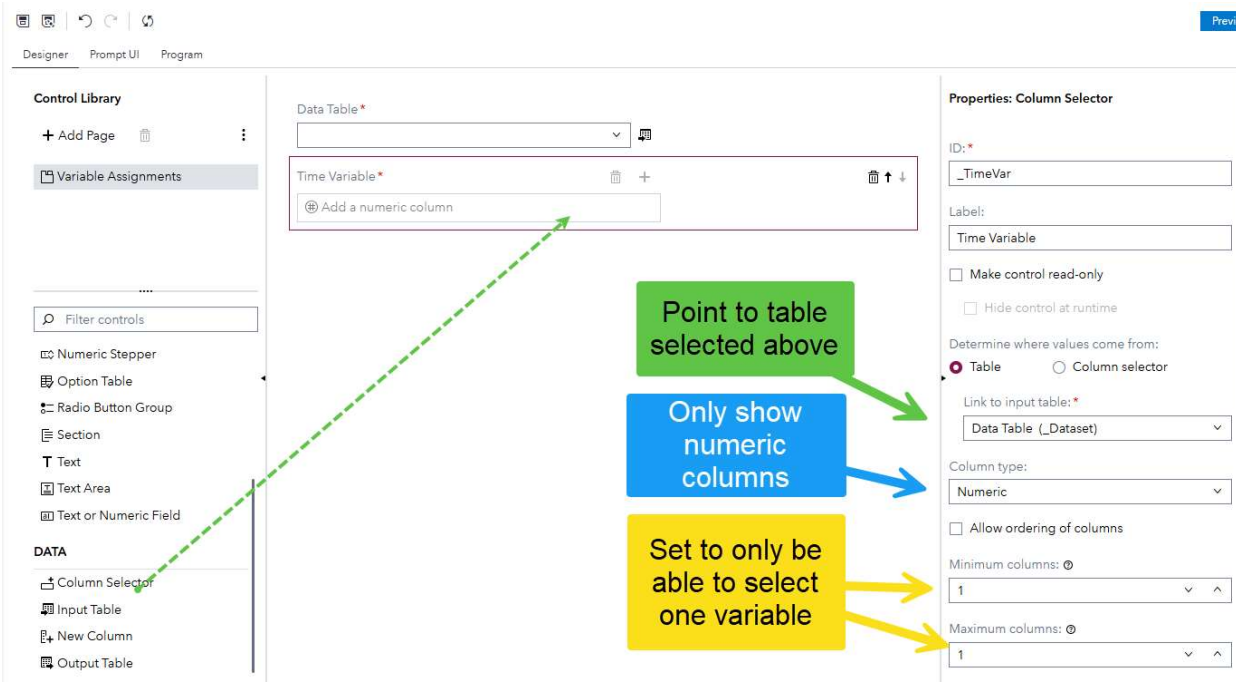

Figure 5: Column Selector

As with the Table Selector, this has an ID and Label to set up. We will use **_TimeVar** and **Time Variable**, respectively. We want this column to be selected from the input table that was already specified, and we want to only show the numeric fields. Finally, we only need one column, so we can make 1 the minimum and maximum for this. Don't forget to save as you go along.

We will repeat this process for the status variable, calling it **_StatusVar,** and making it a numeric selection where only one column is selected. You could have written your macro program to have allowed for either a numeric or a character variable with %IF statements, and allow for either type of value here, but we'll keep it simple and ask for just a number.

Next, we will look into that status variable and find what value is used for censoring, For that, we will utilize a Drop-down list control and set it up as a dynamic list (Figure 6) . We will set this as required (again, you could have written your macro to allow for censoring or not, but we will keep it basic) and select the Status Variable as the column to look at. We'll call this **_CenVal.** We could have provided a list of possible values, but it is much more flexible to allow for dynamic list.

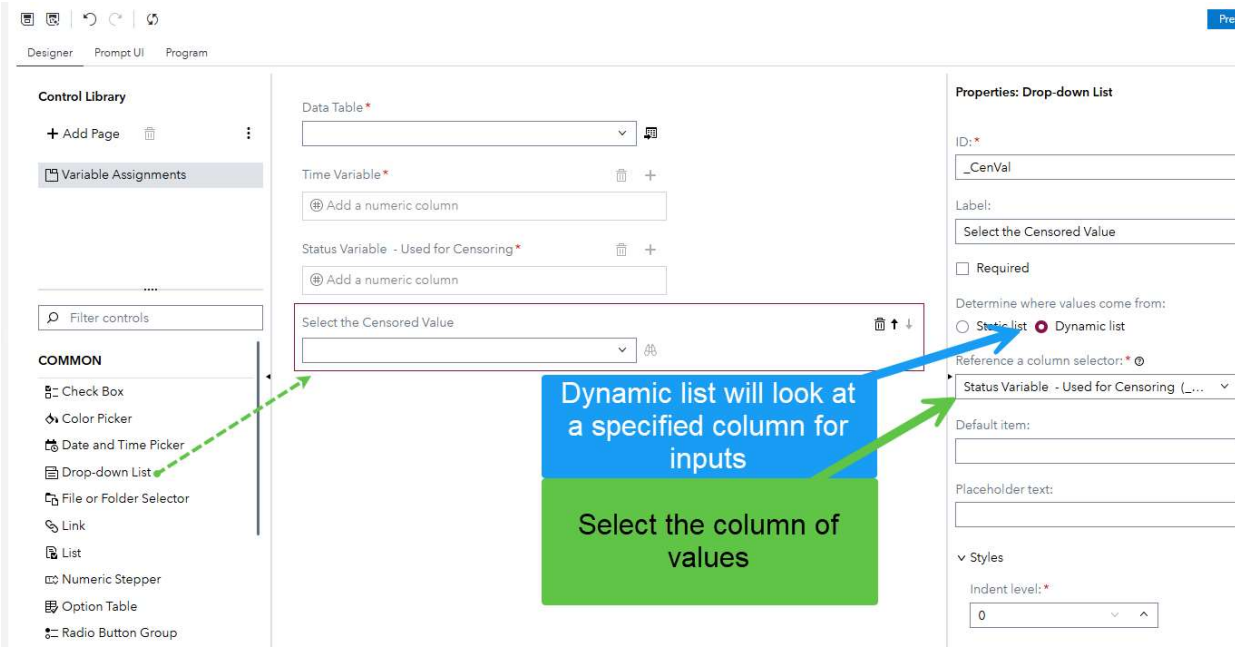

Figure 6: Drop-down List

The last thing we need to do is to capture the stratification variable, which we will do by adding a new column selector for **_GroupVar**. For this variable, we will allow All Types for the column type. That completes our GUI design, and we can see what it looks like by hitting the blue Preview button in the upper right corner of the pallet.

EXECUTING THE STEP

At this point, we are ready to open up the step and test it out. Your step should show up in the Shared tab of the steps menu, go find it there. You can use the filter list to jump right to it. Once you have it, right click on the name and Open in Tab to execute the step (Figure 7).

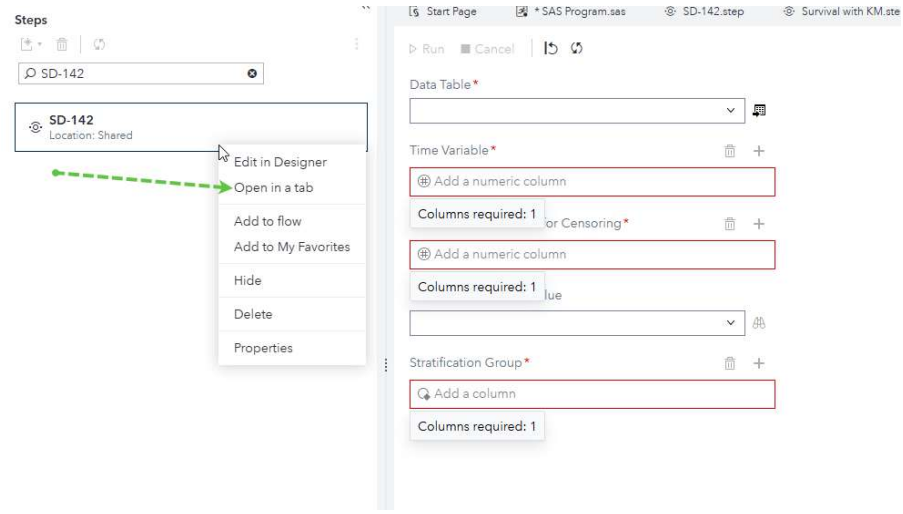

Figure 7: Opening a Step

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.

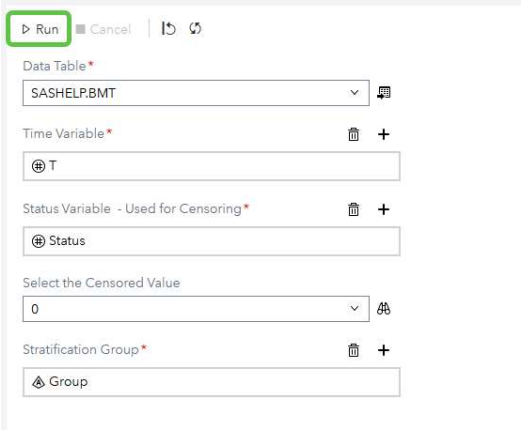

Figure 8: Entering Inputs

We can now run the program and get results. You may have to debug a bit, a common mistake is that you did not assign the IDs correctly to match your macro variables in the program, but if all goes well, you will wind up with output like Figure 9.

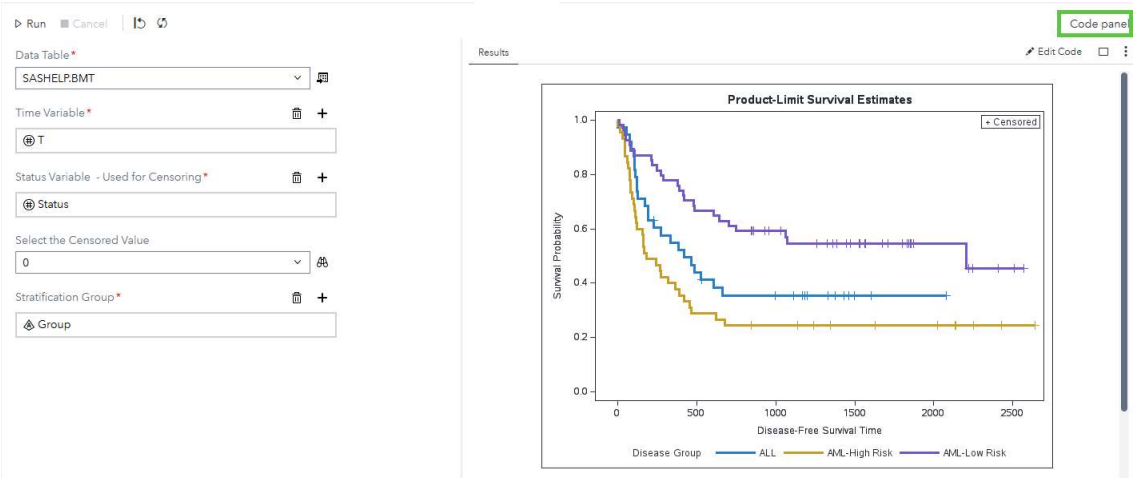

Figure 9: Final Output

By default, the results window will pop up. If you want to examine what happened in the background, you can press the Code Panel button (Figure 9, Green Area) and it will open the results, log, and code tabs for your examination. Do note that there is quite a bit of wrapper code generated, so in this case our 17 lines of code it the Program tab in the designer created 294 lines of executed code.

Custom steps are powerful ways of utilizing macro-based code to share with users who do not want to make changes to the code, but do want to have control over the inputs. You can also take custom steps or existing SAS steps an string them together into a flow for execution.

COMBINING STEPS IN FLOWS

Whether you create your own custom steps or use the ones provided by SAS, you can set up process flows that use several steps to produce repeatable processes that your customer could execute without ever seeing the code. Figure 10 gives an example of this, where we built a flow to make a plot showing the number of adverse events experienced per patient, grouped by treatment.

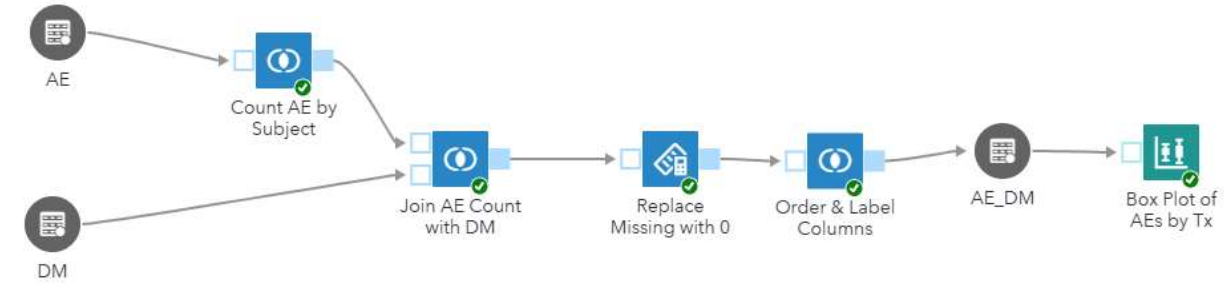

Figure 10: Example Flow

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

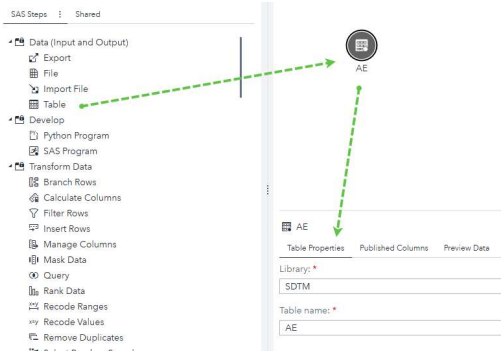

Figure 11: Add a Table

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.

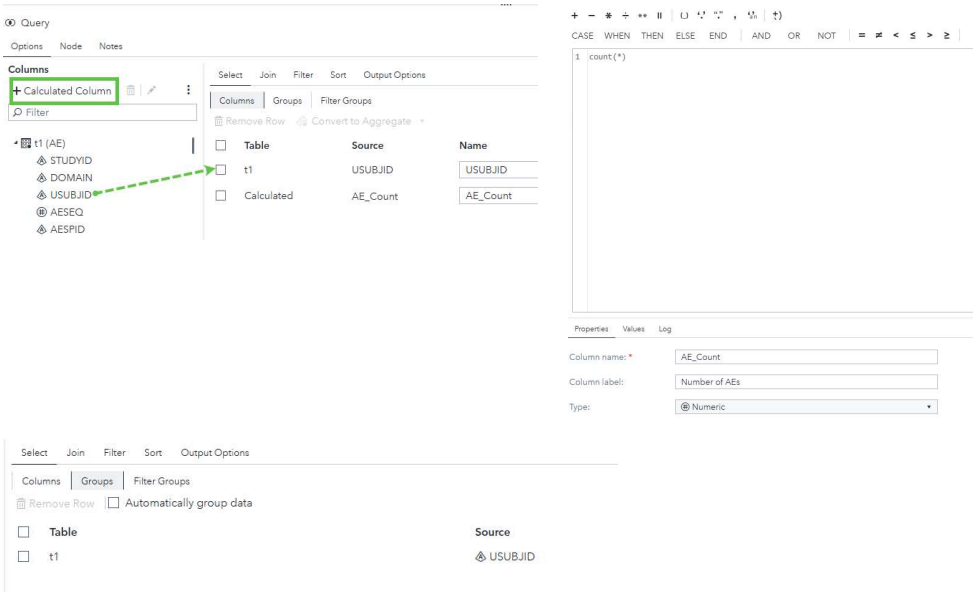

Figure 12: Create a calculated column using the Query step.

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.

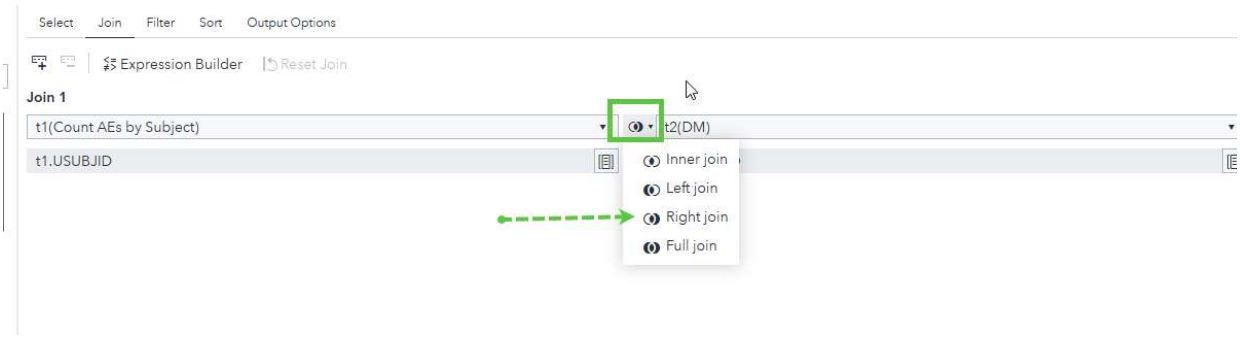

Figure 13: Configuring the Join

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.

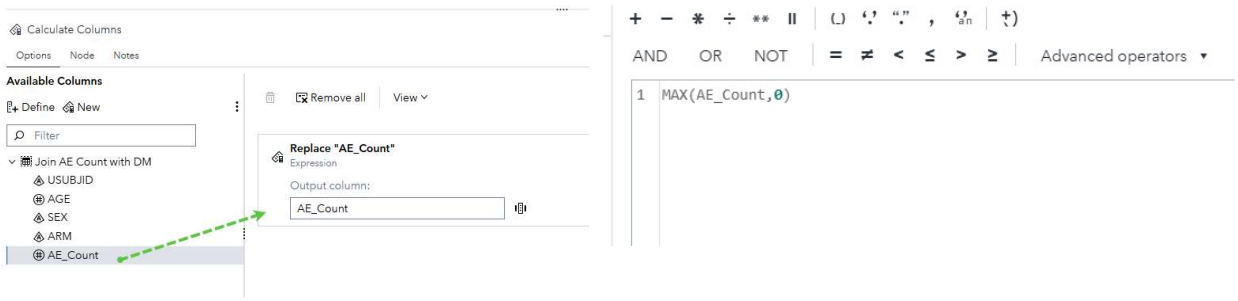

Figure 14: Calculated Column

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.

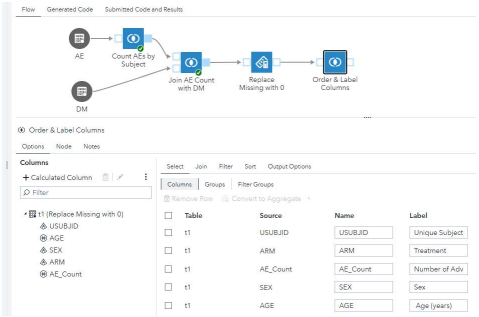

Figure 15: Final column cleanup

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 are an excellent way to make your existing macros into stand-alone objects that non-programmers can interact with and can be deployed across the organization. You can also build out more complicated flows using these custom steps and the steps that SAS provides (and others that are shared out in the programming community). They are a great way to get the most out of all of the macros and other coding assets you already have.

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
- Pritesh did 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:

Jim Box	Pritesh Desai
SAS Institute	SAS Institute
Jim.Box@sas.com	Pritesh.Desai@sas.com