THE SAS DATA STEP

BRIAN CARTER

Dates

asdf

Error in eval(expr, envir, enclos): object 'asdf' not found

# The SAS data step

## Introduction

[I spent most of my career](www.linkedin.com/in/brian-carter-4000522b) as a lonely R programmer in a SAS group that was perpetually failing in its R migration. Changing direction in an organization is always difficult, and in this case the major push back came from non-technical study management staff who still needed SAS for their day to day processes.

These staff managed all of the raw study data coming into our research program. They imported messy questionnaire data from vendors and prepared it for the scientific staff to analyze. The SAS DATA step was their primary domain and they were understandably sensitive to any changes to their workflow that could risk the science.

I often thought that if I could simply translate all their macros, there would be no more excuse and we could cut the cord to SAS, but that strategy was never successful. I came to see that, for large swaths of data professionals, the DATA step was a safe place to do vital technical work. My SAS colleagues took courses in R and after learning the basics deemed the learning curve too steep to make the jump.

This is an entry in a series of blog posts aimed at companies and data professionals who want to make the jump from SAS to R. There are two aspects of the DATA step that promotes a mental model of data processing that make it difficult for SAS users to transition to R. These are the limited data types included in the DATA step and the mechanics of DATA step processing. I hope that fans of the DATA step will find a few strategies to help them along this journey.

## Data types

The SAS DATA Step has exactly two data types: character and numeric. On its face, this seems like it would be a limitation of SAS, but in practice it is a feature. This simplified system includes reasonable defaults for coercion and type conversion, so users don’t often have to worry about the underlying types in their variables. This is in contrast to R where there are 7 atomic variable types that can be expanded and adapted across packages and use cases. The below table roughly maps the basic vector types in R to their corresponding SAS versions.

SAS and R Atomic Types

| SAS | R | Example |
| --- | --- | --- |
| Numeric | Integer | 0L, 1L, 2L, 3L |
|  | Double (numeric) | 1.5, 1e10, -5.9 |
|  | Complex | 1.5i, 3+2i |
|  | Logical | TRUE/ FALSE |
| Character | Character | "I am a character string" |
|  | Raw | 01, ff |

SAS programmers often run into problems when trying to work with these vector types because SAS has incorporated coercion into more of their DATA step processes. SAS programmers don’t often have to think of these coercion processes. Take the following example. This sample data set uses character variables, char1 and char2 as character type, and I want to take a mean of them:

SAS includes a system of formats that allow users to layer additional attributes on these two basic types.

For example, a date variable is stored as a number representing the number of days relative to January 1st, 1960. This provides SAS users the ability to use any numeric operation on a date value, while displaying the results in the desired date format.

The first thing

SAS users often provision a separate formats library to apply a set of formatting to derived variables in a data set. For example, a medical analyst may want to create an indicator variable for disease status in their patients. Binary variables should be coded as ‘0’ and ‘1’, but SAS formats can be used to display these values as ‘No’ and ‘Yes’. This is a powerful feature of SAS that allows users to create derived variables that are easy to interpret.

The DATA step also includes a system of informats that allow users to read in data in a variety of formats. For example, a date variable can be read in as a character string and converted to a date value using the input statement. This system of formats and informats is a powerful feature of SAS that allows users to manipulate data in a variety of ways.

## DATA step mechanics

The SAS DATA step operates on two sequential steps. The DATA step first completes a compilation phase where SAS scans the code for syntax errors, quickly alerting the user to missed semicolons and typos before translating the code into machine language.

SAS then creates the **program data vector (PDV)**, a temporary object where SAS builds a data set, one observation at a time. The PDV includes each of the final variables created by the DATA step with all attributes, including variable type, length, format, and label.

The program is then executed as a loop, with each row of the input data set processed individually and output to the PDV. If no errors are found in the observation, the result is output to a final data set. This process repeats until SAS finishes with the last observation.

Variable attributes are defined at two stages of the DATA step. Users can explicitly code any attribute: character variables can be created for a given length using the INPUT statement, formats and labels can be applied using FORMAT ad LABEL, respectively. Alternatively, SAS can infer these attributes by, typically by the initial or final attributes fed to the PDV.

R and Python users may find this to be a very inefficient process, but it has its benefits. The basic unit of every SAS data frame is the **observation** and this is how SAS users are often thinking of their code. Below are two examples of routine SAS

For example, SAS users would derived a summary variable for each observation by summing three input variables: var = sum(var1, var2, var3);. Since SAS is operating on the observation level, this kind of process makes sense.

Let’s see how this looks in R:

iris |>  
 dplyr::mutate(Sum\_Length = sum(Sepal.Length, Sepal.Width)) |>  
 dplyr::select(Sum\_Length) |>  
 head()

Sum\_Length  
1 1335.1  
2 1335.1  
3 1335.1  
4 1335.1  
5 1335.1  
6 1335.1

So why does R return a single value for each observation of the iris data? The sum() function, like most R functions, is vectorized, meaning that it operates on the entire homogeneous variable at once. This is a more efficient way to process data, but it requires the SAS programmer to transpose their mental model of the data frame. Rather than operating on each observation independently, the R programmer must thing about each variable vector as the fundamental processing unit.

The SAS DATA step splits its process into two independent compilation and executions steps. During compilation, SAS scans the code for syntax errors and then translates it into machine language. It then creates a **Program Data Vector** (PVD) initialized with variable names and attributes to match the desired data output. The program is the executed as a loop, with each row of the input data set being processed and output to the PVD. All variables and their types are determined during compilation; variable attributes are determined when processing the final observation.

On the face of things, this may seem like an inefficient way to process a data set; indeed, many SAS programmers are excited by how much faster R is at a given task, but it does have a few benefits. First, since the input and output data sets exist on disk, SAS programmers can manipulate large data.

## Data types - maybe skip

SAS only includes numeric and character type variables, everything builds off these using formats and attributes. For example, SAS codes dates as time relative to January 1st, 1960. A programmer can treat date values like any other number, adding, subtracting, summarizing, etc. When a SAS programmer uses the mdy() functions or formats a date variable, SAS simply overlays this attribute on a numeric type.

String variables in SAS can be anything, but it important to remember that strings have a length attribute that is defined at various points in the processing, and this often causes problems. Variable length is usually defined by the first string value SAS encounters and will truncate subsequent strings to that length. Length can be explicitly defined in the DATA step, but this requires *a priori* understanding of the data prior to analysis.

When moving to R, SAS users are overwhelmed with the variety of data types associated with modern programming languages. In addition to numeric and character values, R includes integer, logical, complex and raw variables. R developers can use these *atomic* types to create and endless series of object types with associated methods. Missing values may be coded as NA\_real\_ and NA\_character\_ to avoid errors in some applications, but not others.

## Environments

SAS essentially offers two environments for processing workflows. The primary workspace is what you see: DATA and PROCs operate on independent SAS data sets