Mapping SAS Formats to R, Part 1

A Comprehensive Guide for the SAS Programmer

Michael Walshe

# Introduction to SAS Formats

SAS formats and informats are a swiss-army knife of data manipulation, providing in one package methods to:

* Perform a look-up from one value to another
* Group data into bins, based on custom conditions and logic
* Apply a ‘mask’ to data. This means that you can change the way that data is displayed, without changing the underlying data (e.g. a value of -0.6534 could print as (65.3%))
* Read formatted data as a different type, for example to read a value of £1,000.00 as a numeric value

Because of the sheer number of different use-cases for SAS formats, there is no single function or package in R which can fully replace it. However, this can be seen as a benefit, logically separating the different functionalities and making R easier to understand and use. In this article we’ll assess a few different methods for the first two use-cases.

# Look-ups

A SAS format, typically a user-defined one, can be used as a look-up table - for example from a region code to the full name of that region. In R this could be achieved with:

* A join or merge
* A named vector
* Using a function such as dplyr::case\_when to encode the look-up

## Method 1: Using A Join

Joins are a widespread method for performing some form of look-up, we will demonstrate using the base R merge function.

my\_data <- data.frame(  
 region\_code = c("SA", "SA", "E", "AS", "AN", "NA"),  
 measure = runif(6)  
)  
  
lookup\_df <- data.frame(  
 region\_code = c("E", "NA", "SA", "AS", "AF", "AU", "AN"),  
 region\_full = c(  
 "Europe",  
 "North America",  
 "South America",  
 "Asia",  
 "Africa",  
 "Australia",  
 "Antartica"  
 )  
)  
  
merge(my\_data, lookup\_df)

#> region\_code measure region\_full  
#> 1 AN 0.9404673 Antartica  
#> 2 AS 0.8830174 Asia  
#> 3 E 0.4089769 Europe  
#> 4 NA 0.0455565 North America  
#> 5 SA 0.2875775 South America  
#> 6 SA 0.7883051 South America

## Method 2: Using A Named Vector

Where you have a simple mapping of keys to values, a named vector can be a good method in R to move from one to the other. Note here that the keys should be the *names* of the vector, and the values are the elements. Then indexing using a key produces the value.

lookup\_vector <- c(  
 "E" = "Europe",  
 "NA" = "North America",  
 "SA" = "South America",  
 "AS" = "Asia",  
 "AF" = "Africa",  
 "AU" = "Australia",  
 "AN" = "Antartica"  
)  
  
my\_data$region\_full <- lookup\_vector[my\_data$region\_code]

Note that if the keys were numbers, we would have to create the named vector in two steps, first creating a vector of values then assigning to the names function. However this is not recommended, as it can be unclear what e.g. lookup\_vector[3] should return.

## Method 3: Using [{dplyr}](https://dplyr.tidyverse.org/)

dplyr::case\_when() is a very useful way to create a new vector based on several conditions. It is similar to a case when expression in SQL. For a simple look-up, the conditions will be equality with our input vector. This also lets us map values not in our look-up table to some default.

my\_data$region\_full <- dplyr::case\_when(  
 my\_data$region\_code == "E" ~ "Europe",  
 my\_data$region\_code == "NA" ~ "North America",  
 my\_data$region\_code == "SA" ~ "South America",  
 my\_data$region\_code == "AS" ~ "Asia",  
 my\_data$region\_code == "AF" ~ "Africa",  
 my\_data$region\_code == "AU" ~ "Australia",  
 my\_data$region\_code == "AN" ~ "Antartica",  
 .default = "Unknown"  
)

This is a little clunky and includes some repetition, but we can use the more modern case\_match function to improve it, and at the same time move this into a function to let us easily re-use this mapping.

region\_lookup <- function(x) {  
 dplyr::case\_match(  
 x,  
 "E" ~ "Europe",  
 "NA" ~ "North America",  
 "SA" ~ "South America",  
 "AS" ~ "Asia",  
 "AF" ~ "Africa",  
 "AU" ~ "Australia",  
 "AN" ~ "Antartica",  
 .default = "Unknown"  
 )  
}  
  
region\_lookup(my\_data$region\_code)

#> [1] "South America" "South America" "Europe" "Asia"   
#> [5] "Antartica" "North America"

# Binning

For those unaware, “binning” is the process of grouping data in ranges of values. SAS user defined formats can be created to group numeric values into these bins. It’s used in a variety of situations, from analysis and visualisation to statistics. As such, there is a base R function cut that can provide most of the functionality we need.

The simplest behaviour is to provide the number of bins, R will create a factor with labels indicating the intervals.

ages <- c(59, 22, 20, 62, 30)  
  
cut(ages, breaks=3)

#> [1] (48,62] (20,34] (20,34] (48,62] (20,34]  
#> Levels: (20,34] (34,48] (48,62]

We can also define custom intervals and labels, note that labels needs to be 1 element shorter than breaks.

cut(  
 ages,  
 breaks=c(0, 40, 60, 100),  
 labels=c("Young", "Middle-Aged", "Old")  
)

#> [1] Middle-Aged Young Young Old Young   
#> Levels: Young Middle-Aged Old

There are many packages in R that provide a function to bin data for different use-cases, as one example we can again use dplyr::case\_when(), this time to group data. This produces a character vector rather than factor.

dplyr::case\_when(  
 ages <= 40 ~ "Young",  
 ages <= 60 ~ "Middle-Aged",  
 .default = "Old"  
)

#> [1] "Middle-Aged" "Young" "Young" "Old" "Young"

# Honourable Mentions

* A useful package to mention is [{fmtr}](https://fmtr.r-sassy.org/articles/fmtr.html), which is designed to try and replicate the experience of SAS user-defined formats in R, including conditions and format catalogues. However, it also has limited popularity,and it is best practice to try to write idiomatic R rather than to replicate SAS exactly.