



Py R you SAS-ing me?

Converting SAS dataset filters to R and Python

Matt Metherell
phastar



Abstract

The industry is increasingly leaning into open source programming in the likes of R and python, yet most experienced programmers will only be familiar with SAS, and we may also find our guidance defaults to SAS examples.

As a first step in educating SAS-only programmers, we look at how to convert SAS where clauses, which we may find in SDTM sponsor guidance.

- How this is laid out:
- The tables show us the different examples we may find in SAS, and what we want them to be converted to in R and python.
 - The code examples will allow us to convert a SAS statement (saved in an array called *clause*) into the form we need it for R and python.

R is denoted by:  ...and python is denoted by: 

Equal / Not Equal

A common issue that we come across is that SAS has multiple ways to do most tasks, and therefore we see multiple expressions of any concept when receiving SAS code. Therefore we attempt to address as many as possible. In the case of assessing whether a variable and a value are equal or not, there are two options for each case, and the update to both R and python is relatively simple by replacing the comparator.

We have in SAS	We want in R	We want in Python
VAR = "val"	VAR == "val"	VAR == "val"
VAR EQ "val"	VAR == "val"	VAR == "val"
VAR ^= "val"	VAR != "val"	VAR != "val"
VAR NE "val"	VAR != "val"	VAR != "val"

When using = there is not necessarily a space either side, so we need to be mindful of this, and also perform this replacement before the others, or else we may replace != with !=.



```
clause <- gsub("=", '==', clause, ignore.case = TRUE)
clause <- gsub(" EQ ", ' == ', clause, ignore.case = TRUE)
clause <- gsub("^=", '!=', clause, ignore.case = TRUE)
clause <- gsub(" NE ", ' != ', clause, ignore.case = TRUE)
```



```
clause = [c.replace("=", "==") for c in clause]
clause = [c.replace("^=", "!=") for c in clause]
```

To ignore the case of EQ/NE, python needs the re package to be able to ignore the casing.

```
import re

pattern_ne = re.compile(" NE ", re.IGNORECASE)
pattern_eq = re.compile(" EQ ", re.IGNORECASE)

clause = [pattern_ne.sub(" != ", c) for c in clause]
clause = [pattern_eq.sub(" == ", c) for c in clause]
```

Null / Missing

There are many ways to determine if a SAS variable is missing or not. We can use the equal / not equal comparators as shown below to the left, or we can use the missing function. When using EQ/NE SAS will distinguish between numeric and character nulls in a way that R does not, so we have to account for both of these. There are many ways to find null values in R, but for clarity we can use the same option for all cases.

Frustratingly, SAS can use the missing function on both character and numeric variables, which have to be handled differently in python. Therefore, to fully automate this step we need to check the type of variable. Numeric SAS variables will typically be *numpy.float64* when opened in python, and character variables will be *str* assuming that the SAS datasets are read in via the *pyreadstat* package.

We have in SAS	We want in R	We want in Python
VAR = ""	is.na(VAR)	VAR == ""
VAR = .	is.na(VAR)	np.isnan(VAR)
Missing(VAR)	is.na(VAR)	VAR == "" / np.isnan(VAR)

- To avoid repetition on this poster we are:
- assuming equal/not equal comparators have already been replaced
 - not showing examples with both double-quotes and single-quotes
 - not showing the different amounts of spaces that could possibly arise
 - ignoring the negative case (replace not and ^ with a ! early on)

To rearrange the order of the statements, we need to use regular expressions.



```
#deal with eq/ne . and replace with is.na(XXX)
clause <- gsub("[A-Za-z0-9]+) == ' '", 'is.na(\\1)', clause)
clause <- gsub("[A-Za-z0-9]+) == \\. ", 'is.na(\\1)', clause)
```

```
#Update for the missing function
clause <- gsub("not missing\\((([A-Za-z0-9]+)\\))",
"!is.na(\\1)", clause, ignore.case = TRUE)
```

```
clause <- gsub("missing\\((([A-Za-z0-9]+)\\))",
"is.na(\\1)", clause, ignore.case = TRUE)
```



```
import numpy as np
import re

#deal with eq/ne . and replace with isnull()
clause = [c.replace(" == .", ".isnull()") for c in clause]

#replace quote-space-quote with quote-quote
clause = [c.replace("= ' '", "= ''") for c in clause]

#handle cases of the missing function

if type(VAR) == str:
    clause = re.sub(r"missing\\((([A-Za-z0-9]+)\\))",
r"\\1 == ''", clause)

if type(VAR) == np.float64:
    clause = re.sub(r"missing\\((([A-Za-z0-9]+)\\))",
r"np.isnan(\\1)", clause)
```

In / Not In

In clauses are tricky because of the possible inconsistencies in presentation in SAS, in particular whether or not we see commas in between items.

- Python and R both need commas for numerics, SAS does not.
- R needs commas for characters, SAS and python do not.

We have in SAS	We want in R	We want in Python
VAR in (" " "Val1" "Val2")	VAR %in% c("", "Val1", "Val2")	VAR in [" " "Val1" "Val2"]
VAR in (" ", "Val1", "Val2")	VAR %in% c("", "Val1", "Val2")	VAR in [" ", "Val1", "Val2"]
VAR in (. 999 999)	VAR %in% c(NA, 999, 999)	VAR in [999, 999] np.isnan(VAR)
VAR in (., 999, 999)	VAR %in% c(NA, 999, 999)	VAR in [999, 999] np.isnan(VAR)

Like many of the examples on this poster, the simplest solution is to insist on the initial SAS code to be consistent, but in practice this isn't always possible so we have three steps to go through:

- Replacing the in and the open parenthesis.
- Change the closing parenthesis to a square bracket (python only)
- Add commas if they don't exist (complicated – details in paper)



```
#Update in statement
clause <- gsub("([A-Za-z0-9]+) IN \\(", '\\1 %in% c(',
clause, ignore.case = TRUE)
```



```
#Update opening parenthesis
pattern_in = re.compile(" in (", re.IGNORECASE)
clause = [pattern_in.sub(" in [", c) for c in clause]
```

```
#Update closing parenthesis
clause = [c.replace(")", "]") for c in clause]
```

When it comes to adding in commas between items, we can't simply replace quote-space-quote with quote-comma-quote as that could add a comma into a null value.

Similarly, regular expressions are not sufficient as we want to add commas in between, but not at the start or the end, but we also need to make sure we only do this within the parentheses, therefore we need to produce a function that performs the above as well as the following steps:

- Find the word IN
- Find the start and end of the subsequent parentheses
- Within the parentheses if the first character is a quote, then...
 - Count the quotes.
 - Check if a comma exists after an even numbered-quote before the subsequent quote.
 - If missing, add a comma after even-numbered quotes but not the final quote
 - Account for the null option with NA for R and "" for python
- If the first character is a digit, a - sign or ., then...
 - Replace spaces with commas.
 - Account for the null option with NA for R and .isnull() for python

Multiple Clauses

Here we look at combining clauses together by using and/or operators, and one useful thing to note is that **we don't need to change anything for python!**

We have in SAS	We want in R	We want in Python
A and b	A & b	A and b
A & b	A & b	A & b
A or b	A b	A or b
A b	A b	A b

In R we have the almost trivial case of changing words for symbols:



```
clause <- gsub(" AND ", ' & ', clause, ignore.case = TRUE)
clause <- gsub(" OR ", ' | ', clause, ignore.case = TRUE)
```

Final Thoughts

Replace ^ and not with ! early on to avoid having to deal with the negative case of most examples here.

Remember to apply fixes for values using both double-quotes and single-quotes.

Take care when investigating how many spaces we need to deal with.

Beware of null values – this task is much simpler if you are able to disregard them.

