Paper PP08

Py R you SAS-ing Me: Converting SAS dataset filters to R and Python

Matt Metherell, Phastar, Portsmouth, United Kingdom

ABSTRACT

If looking to code SDTMs in R or python, we may still receive sponsor standards that use SAS as their example code for filtering the input datasets, so in this paper I present R code to convert these filters into R, and python code to convert them into python. This can also be used as a ready reckoner for a SAS programmer taking their first steps in R or Python.

INTRODUCTION

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. There are four main changes that we need to make:

1. Equal / Not equal
2. Null / Missing
3. In
4. And / Or

This paper contains sections for each of these four updates, and within each of those sections there is a table that shows us the different examples we may find in SAS, and what we want them to be converted to in R and python. This will be followed by code examples that will allow us to convert a SAS statement (saved in an array called) into the form we need it for R and python.

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.

What we can have in SAS	What we want in R	What 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 !==.

R CODE

To make these replacements in R:

```
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)
```

PYTHON CODE

To make these replacements in python:

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

To treat capital and lowercase versions of EQ/NE the same way, 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]
```

This could be extended to include inequalities, but in practice they rarely arise as dataset filters.

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 above, 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.

What we can have in SAS	What we want in R	What we want in Python
`VAR = " "`	`is.na(VAR)`	`VAR == ""`
`VAR = .`	`is.na(VAR)`	`VAR.isnull()`
`missing(VAR)`	`is.na(VAR)`	`VAR == "" / VAR.isnull()`
`VAR ^= " "`	`!is.na(VAR)`	`VAR != ""`
`VAR ^= .`	`!is.na(VAR)`	`VAR.isnull()`
`not missing(VAR)`	`!is.na(VAR)`	`VAR != "" / VAR.isnull() == False`
`^missing(VAR)`	`!is.na(VAR)`	`VAR != "" / VAR.isnull() == False`

To avoid repetition in this paper we shall:

- assume equal/not equal comparators have already been replaced
- not show examples with both double-quotes and single-quotes
- not show the different amounts of spaces that could possibly arise
- ignore the negative case (ensure to replace not and ^ with a ! early on)
- assume all variable names are in block capitals, as per CDISC standards

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

R CODE

```
#deal with eq/ne null 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("missing\\(([A-Za-z0-9]+)\\)",
       "is.na(\\1) | \\1 == ''", clause, ignore.case = TRUE)
```

PYTHON CODE

```
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.

What we can have in SAS	What we want in R	What 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(NaN, 999, 999)`	`VAR in [999, 999] | VAR.isnull()`
`VAR in (., 999, 999)`	`VAR %in% c(NaN, 999, 999)`	`VAR in [999, 999] | VAR.isnull()`

Like many of the examples on this paper, 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:

1. Replacing the in and the open parenthesis,
2. Change the closing parenthesis to a square bracket (python only)
3. Add commas if they don't exist

R CODE

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

PYTHON CODE

```
#Update open parenthesis
pattern_in = re.compile(" in (", re.IGNORECASE)
[pattern_in.sub(" in [", r) for r in raw]

#Update close parenthesis
raw = [r.replace(")", "]") for r in raw]
```

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 parentheses, therefore we need to produce a function that performs the above as well as the following steps:

1. Find the word IN
2. Find the start and end of the parentheses
3. Within the parentheses if the first character is a quote, then…
    a. Count the quotes.
    b. Check if a comma exists after an even numbered-quote before the subsequent quote.
    c. If missing, add a comma after even-numbered quotes but not the final quote
    d. Replace quote-space-quote with quote-quote for a possible null value.
4. If the first character is a digit, a - sign or ., then…
    a. Replace spaces with commas.
    b. Account for the null option with NaN 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!**

What we can have in SAS	What we want in R	What 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:

R CODE

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

CONCLUSION

This paper outlines the changes required to convert SAS where clauses into R and python. There are four broad replacements that need to be made:

1. Equals / not equals symbols
2. Null values and the missing function
3. In / not in statements
4. And / or conjunctions

The specific tasks required will depend on the state of the instrSome extra things to bear in mind:

- 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.
- R and python are case sensitive for variable names, whereas SAS is not.

In many cases, this task can be simplified, by simplifying the initial SAS code. And with that in mind, it may be useful to use standardised pseudocode comparators, like those used in define.xml to promote cross-platform thinking, and to aid the automation of the define further down the line.

CONTACT INFORMATION

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

Matt Metherell
Phastar
Email: matthew.metherell@phastar.com