

PHUSE EU CONNECT 2023

Paper: CT01

SAS Arrays in R Way

Jagadish Katam

Disclaimer

The following presentation is intended solely for informational purposes. Any content, views, or opinions presented herein are the views of the presenter and not necessarily those of the organization.

Agenda

- Arrays
- SAS vs R
- Examples
- Conclusion
- Q & A

Arrays

- Arrays are useful when we need to perform similar operations or writing similar code on multiple variables.
- SAS arrays provides a simple, efficient way to process a list of variables in a SAS DATA step.
- R provides several ways to work with arrays and multidimensional data structures.

SAS vs R

- We will see how we can do the arrays in R, in comparison with SAS.
- Here is the sample dataset / data frame we are going to use.

x1	x2	x3	x4	x5
1	1	3	AA	AB
2	3	4	AB	CC
3	4	5	AC	DD
4	5	6	AD	EE
5	6	7	AE	FF
6	7	8	AF	GG

Example 1: Numeric variables having value greater than 3 need to be replaced with missing value

SAS

```
data test;  
set temp;  
array var1 (3) _numeric_;  
do i = 1 to dim(var1);  
if var1{i}>3 then var1{i}=.;  
end;  
run;
```

R

Approach 1:

```
test <- temp %>% mutate(across(x1:x3, ~  
ifelse(>3, NA_real_, .)))
```

Approach 2:

```
var <- names(which(sapply(temp,  
is.numeric)))
```

```
test <- temp %>%  
mutate(across(all_of(var), ~ ifelse(>3,  
NA_real_, .x)))
```

Example 1: Code Review

- ``across(x1:x3, ~ ifelse(>3, NA_real_, .))``: This part of the code is using the ``across()`` function, which applies the specified operation to multiple columns.
- ``var <- names(which(sapply(temp, is.numeric)))``: This line extracts the names of numeric variables in the 'temp' data frame.
- ``sapply(temp, is.numeric)`` checks which columns in the 'temp' data frame are of numeric data type. It returns a logical vector with ``TRUE`` for numeric columns and ``FALSE`` for non-numeric ones.
- ``which()`` is used to find the indices where the logical vector is ``TRUE``, corresponding to numeric columns.

Example 1: Output

SAS

- Missing values are represented by a period (".") in numeric variables.

x1	x2	x3	x4	x5
1	1	3	AA	AB
2	3		AB	CC
3			AC	DD
			AD	EE
			AE	FF
			AF	GG

R

- Missing values are represented by the special value "NA".
- This applies to both numeric and character variables.

x1	x2	x3	x4	x5
1	1	3	AA	AB
2	3	NA	AB	CC
3	NA	NA	AC	DD
NA	NA	NA	AD	EE
NA	NA	NA	AE	FF
NA	NA	NA	AF	GG

Example 2: Extract first letter of all the character variables

SAS

```
data test;
set temp;
array cvars (*) _character_;
do i = 1 to dim(cvars);
cvars{i} = substr(cvars{i},1,1);
end;
drop i;
run;
```

R

```
varc <- names(which(sapply(temp,
is.character))))
```

```
test <- temp %>%
mutate(across(all of(varc),
~stringr::str_sub(.x,1,1)))
```

Example 2: Output & Code Review

OUTPUT

x1	x2	x3	x4	x5
1	1	3	A	A
2	3	4	A	C
3	4	5	A	D
4	5	6	A	E
5	6	7	A	F
6	7	8	A	G

- ``~stringr::str_sub(.x, 1, 1)``: This is the function that is applied to each column in the 'varc' selection.
- ``stringr::str_sub(.x, 1, 1)`` takes the first character (letter) from the input column represented by ``.x``.
- the ``1, 1`` arguments specify the starting and ending positions for the substring, effectively extracting only the first letter.

Example 3: Assign initial values in an Array

SAS

```
data test;
set temp;
array nvars (*) _numeric_;
array pctinc {3} _temporary_ (1, 2 , 3);

do i = 1 to dim(nvars);
nvars{i} = nvars{i} * pctinc{i};
end;
drop i;
run;
```

R

Vector:

```
val <- c(x1=1, x2=2, x3=3)
```

Approach 1:

```
test <- temp %>% mutate(across(all_of(var),
~.x[val[cur_column()]]))
```

Approach 2:

```
temp1 <- purrr::map2_df(temp[,var], val, `*`)
test <- cbind(temp1, temp[,c('x4', 'x5')])
```

Example 3 : Output & Code Review

Output

x1	x2	x3	x4	x5
1	2	9	AA	AB
2	6	12	AB	CC
3	8	15	AC	DD
4	10	18	AD	EE
5	12	21	AE	FF
6	14	24	AF	GG

- `~.x*val[cur_column()]`: this is the function that is applied to each column in 'var'.
- `.x` represents the values in each column.

- `val[cur_column()]` is used to fetch the corresponding value from the 'val' vector.
- `cur_column()` returns the current column name, and this is used to index the 'val' vector. for example, if the current column name is 'x1', it will fetch the value 1 from 'val'.
- `map2_dfc()`: this function is used to apply this element-wise multiplication to all columns and returns a data frame as the result.'
- `cbind()` combines the two data frames, with 'temp1' on the left and the selected 'x4' and 'x5' columns on the right.

Example 4: Find the minimum value from a list of variables in an Array

SAS

```
data test;
set temp;
array nvars (*) _numeric_;
do i = 1 to dim(nvars);
minval = min(of nvars{*});
end;
drop i;
run;
```

R

```
test <- temp %>% rowwise() %>%
mutate(minval=min(across(all_of(var)
)))
```

Example 4 : Output & Code Review

Output

x1	x2	x3	x4	x5	minval
1	1	3	AA	AB	1
2	3	4	AB	CC	2
3	4	5	AC	DD	3
4	5	6	AD	EE	4
5	6	7	AE	FF	5
6	7	8	AF	GG	6

- ``rowwise()`` function is applied, which indicates that the following operations will be performed row-wise.
- the ``min()`` function is then applied to these selected columns to calculate the minimum value for each row.

Example 5: Find the sum of values from a list of variables in an Array

SAS

```
data test;  
set temp;  
array nvars (*) _numeric_;  
do i = 1 to dim(nvars);  
sum = sum(of nvars{*});  
end;  
drop i;  
run;
```

R

```
test <- temp %>%  
mutate(sum=rowSums(apply(all_of(var  
))))
```

Example 5 : Output & Code Review

Output

x1	x2	x3	x4	x5	sum
1	1	3	AA	AB	5
2	3	4	AB	CC	9
3	4	5	AC	DD	12
4	5	6	AD	EE	15
5	6	7	AE	FF	18
6	7	8	AF	GG	21

- `rowSums()`: the ``rowSums()`` function is applied to calculate the sum of values across the selected columns for each row in the data frame.

Example 6: Compare one set of columns with another set of columns

SAS

```
data temp;
set temp;
array var1 (3) jan feb mar;
array var2 (3) apr may jun;
array var3 (3)$ apr1 may1 jun1;
do i = 1 to dim(var1);
do j = 1 to 3;
if var1{i} = var2{j} then var3{i}='Y';
end;
end;
flag=coalescec(of var3{*}, 'N');
drop apr1 may1 jun1 i j;
run;
```

R

```
temp$flag <- sapply(1:nrow(temp),
\ (x)
ifelse(any(unlist(temp[x, 1:3]) %in%
unlist(temp[x, 4:6])), "Y", "N"))
```

jan	feb	mar	apr	may	jun
10	20	30	5	0	10
7	9	6	5	9	8
1	2	3	4	5	6

Example 6 : Output & Code Review

Output

jan	feb	mar	apr	may	jun	flag
10	20	30	5	0	10	Y
7	9	6	5	9	8	Y
1	2	3	4	5	6	N

- ``sapply(1:nrow(temp), \(x) ...``: this section applies a function to each row in the 'temp' data frame.
- ``unlist(temp[x, 1:3])`` extracts and unlists the values from columns 1 to 3 for the current row 'x'.
- ``unlist(temp[x, 4:6])`` extracts and unlists the values from columns 4 to 6 for the same row 'x'.
- - ``ifelse(..., "Y", "N")`` assigns "Y" if there is at least one common element (the condition is true), and "N" if there are no common elements.

Conclusion

- Both SAS and R provide array functionality that is essential for data manipulation and analysis.
- SAS is preferred in enterprise environments with a budget for licensing.
- R is popular among data scientists, statisticians, and academics due to its open-source nature and extensive statistical capabilities.
- So the choice should align with the project requirements and the tools the team is most comfortable using.

Thank you

Any Questions?

Jagadish Katam

Princeps Technologies

Email: Jagadish.katam@princepstechnologies.com

LinkedIn: [Jagadish Katam | LinkedIn](#)