Paper CT03

R String Manipulation Functions vs SAS Character Functions

Jagadish Katam, Princeps Technologies Inc.

ABSTRACT

In this paper will do a comparative analysis of character functions between R and SAS. This is a better way of learning any new programming language. The industry has been using SAS for quite some time and recently there is a new trend of R in clinical trials. Considering the same, we might soon move on to use R in generating datasets and outputs, so for data manipulation we need to know how and which R string function we need use for the data manipulation. Some of the common R string functions include paste(), str_replace(), str_split(), str_subset() etc., including the regex functions like str_view() which we will try to understand and compare them with SAS character functions like catx(), tranwrd(), scan(), find() and prxmatch(). We will explore the string functions of R vs SAS with some examples. This paper will help programmers to understand R string functions.

INTRODUCTION

In this paper we will cover some of the many R string manipulation functions, and compare these to SAS character functions. Basic string manipulation typically includes case conversion, simple character, abbreviating, substring replacement, adding/removing whitespace, and performing set operations to compare similarities and differences between two character vectors. These operations can all be performed with base R functions; however, some operations (or at least their syntax) are greatly simplified with the stringr package. This paper illustrates functions such as paste() and strsplit() which are part of base R string manipulation, and str_detect(), str_extract(), str_replace() and str_view_all() which are part of stringr package.

stringr PACKAGE

The stringr package was developed by Hadley Wickham to act as simple wrappers that make R's string functions more consistent, simple, and easier to use. It is part of the core tidyverse. To use these functions, you will need to install and load the stringr package:

```
# install tidyverse package
install.packages("tidyverse")

# install stringr package
install.packages("stringr")

# load package
library(tidyverse)
library(stringr)
```

1. str_detect() vs find()

str_detect() function in R Language is used to check if the specified match of the substring exists in the original string. It will return TRUE for a match found otherwise FALSE against each of the element of the Vector or Matrix. In a data frame or dataset you need to use filter() function to output the TRUE records. Similarly in SAS, you can use find() function along with the where statement.

Note: This function uses 'stringr' Library.

Syntax: str_detect(string, pattern)

Parameter:
string: specified variable (string)
pattern: word (Pattern) to be matched

Example 1.1:
Consider the sample data with concomitant medications (CONMED).

CONMED
PILSICAINIDE
PHENOBARBITAL
BUTALBITAL
RIFAP

To extract the patients who took a concomitant medication like “RIFAP”. You need to use str_detect() function within filter() function to subset rows in the dataset CM. CM2 is the new dataset created using the CM dataset from str_detect() function.

R

```
CM2 <- filter(CM,str_detect(CONMED,"RIFAP"))
```

SAS

```
data CM2;
   set CM;
   where find(CONMED,"RIFAP");
run;
```

Output:

CONMED
RIFAP

2. paste() vs catx()

paste() method in R programming is used to concatenate the two string values by separating with delimiters. The default delimiter in paste() function is space. To concatenate the variable values/string values with a delimiter, you need to use the sep= argument/parameter in paste(). In SAS, you can use the catx() function to concatenate the variable values with a delimiter.

Note: This function is part of base R.

Syntax: paste(string1, string2, sep=)

Return: Returns the concatenate string.

Example 2.1:
Let’s see an example to concatenate the CONMED and DOSE into a new variable CONMEDOS with space.

CONMED	DOSE
ZOMIG	5 mg
TYLENOL	1000 mg
CLAVULANIC ACID	62.5 mg

R

```
CM$CONMEDOS <- paste(CM$CONMED,CM$DOSE)
```

SAS

```
data CM;
   set CM;
   CONMEDOS=catx(" ", CONMED,DOSE);
run;
```

Output:

CONMED	DOSE	CONMEDOS
ZOMIG	5 mg	ZOMIG 5 mg
TYLENOL	1000 mg	TYLENOL 1000 mg
CLAVULANIC ACID	62.5 mg	CLAVULANIC ACID 62.5 mg

Example 2.2:
Let's see another example to concatenate the CONMED and DOSE into a new variable CONMEDOS with a delimiter "/".

R

```
CM$CONMEDOS <- paste(CM$CONMED,CM$DOSE,sep="/")
```

SAS

```
data CM;
   set CM;
   CONMEDOS=catx("/", CONMED,DOSE);
run;
```

Output:

CONMED	DOSE	CONMEDOS
ZOMIG	5 mg	ZOMIG/5 mg
TYLENOL	1000 mg	TYLENOL/1000 mg
CLAVULANIC ACID	62.5 mg	CLAVULANIC ACID/62.5 mg

2.1 paste0()

paste0() method in R programming is used to concatenate all elements without separator. Hence, in the example below you can see the CONMED and DOSE are concatenated without a space whereas the same when done with paste() function concatenates with a space.

Example 2.1.1:

R

```
CM$CONMEDOS <- paste0(CM$CONMED,CM$DOSE)
```

Output:

CONMED	DOSE	CONMEDOS
ZOMIG	5 mg	ZOMIG5 mg
TYLENOL	1000 mg	TYLENOL1000 mg
CLAVULANIC ACID	62.5 mg	CLAVULANIC ACID62.5 mg

2.2 CAVEAT TO USING paste()

The challenge in using the paste() function with a delimiter when compared with catx in SAS is that, when the values are missing in either or both variables used for concatenation, then paste() will still display the delimiter "/", whereas catx() does not.

Example 2.2.1:

R

```
CM$CONMEDOS <- paste(CM$CONMED,CM$DOSE,sep="/")
```

Output:

CONMED	DOSE	CONMEDOS
ZOMIG	5 mg	ZOMIG/5 mg
TYLENOL		TYLENOL/
CLAVULANIC ACID	62.5 mg	CLAVULANIC ACID/62.5 mg

SAS

```
data CM;
   set CM;
   CONMEDOS=catx("/", CONMED,DOSE);
run;
```

CONMED	DOSE	CONMEDOS
ZOMIG	5 mg	ZOMIG/5 mg
TYLENOL		TYLENOL
CLAVULANIC ACID	62.5 mg	CLAVULANIC ACID/62.5 mg

2.3 HOW TO OVERCOME THE CHALLENGE WITH paste()

You can use the ifelse() function to remove the "/" if DOSE is missing as done in the below R code. This is an alternate approach to overcome the challenge posed by paste() function. Note there are multiple ways to deal with this issue, this is just one solution.

Example 2.3.1:

R

```
CM$CONMEDOS <-
paste(CM$CONMED,
ifelse(CM$DOSE == " ", "", "/"),CM$DOSE)
```

Output:

CONMED	DOSE	CONMEDOS
ZOMIG	5 mg	ZOMIG / 5 mg
TYLENOL		TYLENOL
CLAVULANIC ACID	62.5 mg	CLAVULANIC ACID / 62.5 mg

3. strsplit() vs scan()

The strsplit() in R programming language function is used to split the elements of the specified character vector into substrings according to the given substring taken as its parameter.

Note: This function is part of base R.

Syntax: strsplit(x, split)

Parameters:
x: This is the character vector, data file, or a string who's each element is going to be split.
split: This parameter says to split the specified string X into the required formats.

Return value: This function returns a new split string.

Example 3.1:
In order to split the CONMEDOS into CONMED and DOSE we can use the strsplit() function whereas in SAS you can use scan() function.

R

```
CM$CONMED <- sapply(strsplit(as.character
        (CM$CONMEDOS), "/"),"[", 1)


CM$DOSE <- sapply(strsplit(as.character
        (CM$CONMEDOS), "/"),"[", 2)
```

SAS

```
data CM;
   set CM;
   CONMED=scan(CONMEDOS,1,"/");
   DOSE= scan(CONMEDOS,2,"/");
run;
```

Output:

CONMEDOS	CONMED	DOSE
ZOMIG/5 mg	ZOMIG	5 mg
TYLENOL/1000 mg	TYLENOL	1000 mg
CLAVULANIC ACID/62.5 mg	CLAVULANIC ACID	62.5 mg

as.character()

as.character() function in R Language is used to convert a numeric object to character object.

sapply()

sapply() function in R Language takes list, vector or data frame as input and gives output in vector or matrix. It is useful for operations on list objects and returns a list object of same length of original set.

Syntax: sapply(X, FUN)

Parameters:
X: A vector or an object
FUN: Function applied to each element of x

You can use the strsplit() only on character columns, so need to convert it first to character and then using the strsplit() within the sapply() and you can extract the value using the '[' and column number (1 or 2 ..) into different columns/variables.

4. str_extract() vs substr()

The str_extract() in R programming language function is used to extract matching patterns from a string.

Note: This function uses 'stringr' Library.

Syntax: str_extract(string, pattern)

Parameters:
String: Input character vector.
pattern: The default interpretation is a regular expression.

The str_extract() function uses the regular expression (metacharacters) \\d+\\s\\w+ or \\d+\\.\\d\\s\\w+ to extract the dose with digits with or without a dot followed by space and letters. In SAS, you can use findc() function to detect the position of the digit and then use substr() to extract the dose.

Example 4.1:

R

```
CM$DOSE <- str_extract(CM$CONMEDOS,
"\\d+\\s\\w+|\\d+\\.\\d\\s\\w+")
```

SAS

```
data CM;
   set CM;
   DOSE=substr(CONMEDOS,findc(CONMEDOS,,"d"));
run;
```

Output:

CONMEDOS	DOSE
ZOMIG/5 mg	5 mg
TYLENOL/1000 mg	1000 mg
CLAVULANIC ACID/62.5 mg	62.5 mg

5. str_replace() vs tranwrd()

The str_replace() function only replaces the first match. In SAS, you can use the tranwrd() function to replace all the string, n number of time where it matches.

Note: This function uses 'stringr' Library.

Syntax: str_replace(string, pattern, replacement)

Parameters:
string: A vector or an object
pattern: The default interpretation is a regular expression.

Example 5.1:
In the example below, we are using the str_replace() in R to replace the "mg" with "mcg". In SAS, tranwrd() does the same. But the issue with both these functions is that they will replace "mg" with "mcg" if it happens to be there in the middle of the word as well. So, need to be careful while using these functions. Alternatively, the advantage with str_replace() in R is that we can use regular expressions which recognize the pattern to replace.

R	Output:
`CM$CONMEDOS <- str_replace(CM$CONMEDOS, "mg", "mcg")`	
SAS	

`data CM;`
`  set CM;`
`  CONMEDOS=tranwrd(CONMEDOS , "mg", "mcg");`
`run;`	

5.2 Sample data

CONMED
MULTIVITAMIN /00831701/
TYLOX /00446701/
WOBENZYM N /01199401/

Example 5.2:
In the example below, we are using the str_replace() in R to replace the unnecessary data with "//".

R	Output:
`CM$CMDECOD <-` `str_replace(CM$CMDECOD, "/00831701/", " ")`	

5.2 str_replace() vs prxchange() with regular expression

The regular expression "\\/\\d+\\/ used in str_replace() locates the text which starts with "/" followed by digits and ends with "/" and replace with space. In SAS, you can use the same regular expression in prxchange() function. The difference between the usage of metacharacters between R and SAS is that whenever a "\" appears in a SAS regular expression, you must write it as "\\" in the R regular expression.

Example 5.2.1:

R	Output:
`CM$CONMED <- str_replace(CM$CONMED, "\\/\\d+\\/", " ")`	
SAS	

`data CM;`
`  set CM;`
`  CONMED=prxchange("s/\/\d+\/ / /",-1, CONMED);`
`run;`	

6. str_view_all()

str_view() and str_view_all() create HTML widgets that display regular expression matches. str_view() shows the first match only; str_view_all() shows all the matches. Both are in the stringr package.

To see the HTML view in the viewer window, you need to install and load the below package

```
# install htmlwidgets package
install.packages("htmlwidgets")

# load package
library(htmlwidgets)
```

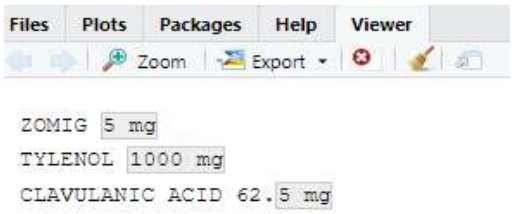
Syntax:
str_view(string, pattern, match = NA)
str_view_all(string, pattern, match = NA)

Parameters:
string: Input character vector.
pattern: The default interpretation is a regular expression.
match: If TRUE, shows only strings that match the pattern. If FALSE, shows only the strings that don't match the pattern. Otherwise (the default, NA) displays both matches and non-matches.

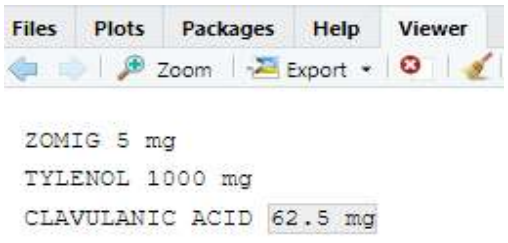
Regular expression is a very compact and powerful language that allows you to describe patterns in strings. They are nothing but a sequence of metacharacters that matches a pattern in a piece of text or a text file. The metacharacters of the regular expression are pretty similar in all the languages. But the functions of extracting, locating, detecting, and replacing can be different in different languages. Like in SAS the metacharacters are represented with "\" whereas in R they are represented with "\\".

Here we'll use str_view_all() function which uses a character vector and a regular expression to demonstrate how they match in the output. str_view_all() will not create new variables, but is a helper to check if the regular expression pattern which is used matches the text or not. So, it is a regular expression pattern checker. This str_view_all() function will print the resulting data in the viewer window and highlight the pattern that matches the regular expression used.

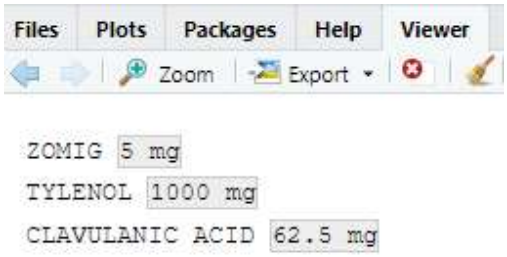
Example 6.1:
The regular expression \\d+\\s\\w+ used in the below code matches the digits followed by space and letters, due to which it does not match 62.

R	Output:
`str_view_all(CM$CONMEDOS,"\\d+\\s\\w+",match=TRUE)`	

Example 6.2:
The regular expression with \\d+[\\.]\\d+\\s\\w+ to match the dose with dot as well.

R	Output:
`str_view_all(CM$CONMEDOS,` `"\\d+[\\.]\\d+\\s\\w+",match=TRUE)`	

Example 6.3:
In the below regular expression, you can combine the above 2 regular expression separated by pipe "|" which acts as "or" and matches both the patterns.

R	Output:
`str_view_all(CM$CONMEDOS, "\\d+[\\.]\\d+\\s\\w+|\\d+\\s\\w+",match=TRUE))`	

CONCLUSION

In this paper I tried to compare the R string functions with SAS character function. The best way to learn any new programming language is to learn by comparison. Since most of the clinical SAS programmers are already familiar with SAS functions, so working with similar functions in R could help to learn R easily. There are a lot of R string functions but we discussed only a few important functions. All the outputs shown in this paper are from R, as most of us already know how it looks in SAS. Apart from couple of Base R functions, we also discussed the stringr package as it provides an easy-to-use toolkit for working with strings, i.e., character data, in R. Thus, this paper hopes to guide you through Base R/stringr package functions for manipulating strings along with a concise reference to regular expressions, a mini-language for describing, finding, and matching patterns in strings.

REFERENCES

Geeks For Geeks. [Online] Available from: R Programming Language - Introduction - GeeksforGeeks

R Notebook to Practice R. Retrieved from https://www.kaggle.com/

Chapter 11 Strings with stringr. Retrieved from R For Data Science (rstudio-pubs-static.s3.amazonaws.com)

Work with strings with stringr : : CHEAT SHEET. Retrieved from strings (evoldyn.gitlab.io)

ACKNOWLEDGMENTS

The author would like to thank Jessica Whittaker-Dixon (Stream Chair), John van Bemmelen (Stream Chair), Bharat Buchupalli, Kishor Badikala and Vijay Krishna for their insightful comments and review.

RECOMMENDED READING

A very good introductory book for R is "R in action", Second edition (Early Access Edition, 2013) by Robert Kabacoff.

For a comparison between SAS and R with a large number of side-by-side code examples please have a look at "SAS and R", Second Edition (2014) by Ken Kleinman and Nicholas J. Horton.

CONTACT INFORMATION

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

Jagadish Katam
Princeps Technologies Inc.
Jagadish.katam@princepstechnologies.com