datasailr - An R Package for Row by Row Data Processing, Using DataSailr Script

by Toshihiro Umehara

Abstract Data processing and data cleaning are essential steps before applying statistical or machine learning procedures. R provides a flexible way for data processing using vectors. R packages also provide other ways for manipulating data such as using SQL and using chained functions. I present yet another way to process data in a row by row manner using data manipulation oriented script, DataSailr script. This article introduces datasailr package, and shows potential benefits of using domain specific language for data processing.

Introduction

For data analysis and statistics, processing data is a very important step before applying statistical procedures or machine learning procedures. R as a programming language provides a very flexible way of data processing using vectors. There are also R packages that help users to manipulate data in different ways, such as **sqldf**(Grothendieck, 2017) and **dplyr**(Wickham et al., 2021). **sqldf** enables users to write SQL for data manipulation. **dplyr** enables users to write data manipulation procedures in a sequential way by chaining functions, which has a great compatibility with forward-pipe operators %>%[1] and |>[2].

Different from R, SAS[3] software (SAS Institute, 1985), as one of the most famous commercial statistical systems, provides another way to manipulate data. SAS software provides DATA blocks, within which users can write a script that is specific for data manipulation in a row wise manner. The separation of data manipulation steps and statistical procedures has a benefit to focus on each step and improved readability. Its row wise data manipulation script is easy to understand and learn.

This paper describes the **datasailr** package, the package that enables row by row data processing, using DataSailr script. I will explain how to use this package, the grammar of DataSailr script and its limitations. Also, there is another aim of this paper. As mentioned about SAS software's DATA block, I would like to emphasize potential benefits of using domain specific language for data processing by introducing this package.

Motivation

DataSailr, which package name is **datasailr**, brings intuitive row by row data manipulation to R. The data manipulation instructions for each row is written in DataSailr script, which is an easy script designed especially for data manipulation. In default R, data frames are manipulated using column vectors and vector operations. It depends on each user's preference and objective, but row wise approach can reduce complexity compared to vector calculation.

A famous R package, **dplyr**, has been improving the same kind of points. It enables data manipulation without thinking much about column vectors. Pipe operators, %>% or |>, and **dplyr** functions realize intuitive data manipulation flow. DataSailr enables the same kind of things with a single DataSailr script.

How DataSailr Works

Applying DataSailr Script to Data Frame

DataSailr has a main function called sail() function. It takes data frame and DataSailr script, and the data frame is processed following the script. The first argument is data frame, and can work with pipe operators. The second argument is DataSalr script, and each row is processed following this script as described in Figure 1. UTF-8 encoding is strongly encouraged for both DataSailr scripts and input

[1]Forward-pipe operator was developed in **magrittr**(Bache and Wickham, 2020)

[2]This pipe operator is introduced in a developing version of R (`https://developer.r-project.org/blosxom.cgi/R-devel/NEWS/2021/01/15#n2021-01-15`)

[3]SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries.

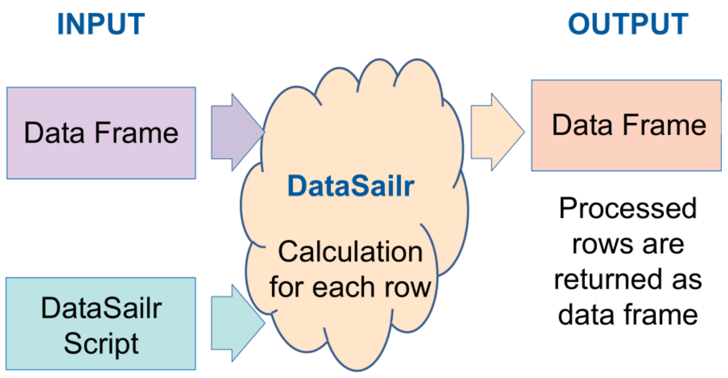

Figure 1: How DataSailr Processes Data

data frame because some built-in functions and regular expression assume UTF-8 encoding for their behaviors. Also, in writing DataSailr script, using raw character constant literal[4] `r"(...)"` can help when the script contains backslashes which are interpreted as parts of escape sequences of R string.

Example of DataSailr Script

The following example script conducts data processing for R's built-in 'mtcars' data. The example script generates description, country and manufacturer columns for each car using hp and mpg columns and row names. The script consists of variable assignment, if-else statements, special variables, regular expression matching, backreference and built-in functions. These topics are explained in the section of 'Grammar of DataSailr Script'.

```
library(datasailr)
data(mtcars)

result = datasailr::sail( mtcars, '
  // Comments in DataSailr start with double slashes
  // if-else statement: Check if hp column value is larger than 150 or not.
  if(hp > 150 ){
    // Variable assignment: powerful column value is set "powerful"
    powerful = "powerful "
  }else {
    powerful = "normal power "
  }

  // if-else can also be written like this
  if(mpg > 20){ efficient = "efficient " }
  else {  efficient = "" }

  // Regular expressions that match manufacturer names
  // Assigning regular expressions to variables do not affect data frame
  // , but can become reused at different lines.
  germany = re/(^Merc|^Porsche|^Volvo)/
  japan = re/(^Mazda|^Honda|^Toyota)/
  datsun = re/(^Datsun)/
  hornet = re/(^Hornet)/
  valiant = re/(^Valiant)/
  duster = re/(^Duster)/

  // Regular expression matching with special variable _rowname_
  if ( _rowname_ =~ germany ) {
    country = "Germany"
    // Backreference to matched strings
    // rexp_matched(1) means the first grouped sub string.
```

[4]available since R 4.0.0. Details can be seen by `?Quotes`

```
    manufacturer = rexp_matched(1)
  }else if ( _rowname_ =~ japan ) {
    // semicolons can be used as terminals of statements
    country = "Japan"; manufacturer = rexp_matched(1)
  }else if ( _rowname_ =~ datsun ) {
    country = "Japan"; manufacturer = "Nissan"
  }else if ( _rowname_ =~ hornet ) {
    country = "USA"; manufacturer = "AMC"
  }else if ( _rowname_ =~ valiant ) {
    country = "USA"; manufacturer = "Chrysler"
  }else if ( _rowname_ =~ duster ) {
    country = "France"; manufacturer = "Renault"
  }else{
    country = ""
    manufacturer = ""
  }

  // built-in function str_concat()
  desc = str_concat( "", powerful, efficient, "car")
' )

# show the first 10 rows of result data frame.
print( head( result[c( "hp", "mpg", "desc", "country", "manufacturer")] , 10 ))
```

```
                   hp  mpg                       desc country manufacturer
Mazda RX4         110 21.0 normal power efficient car   Japan        Mazda
Mazda RX4 Wag     110 21.0 normal power efficient car   Japan        Mazda
Datsun 710         93 22.8 normal power efficient car   Japan       Nissan
Hornet 4 Drive    110 21.4 normal power efficient car     USA          AMC
Hornet Sportabout 175 18.7               powerful car     USA          AMC
Valiant           105 18.1           normal power car     USA     Chrysler
Duster 360        245 14.3               powerful car  France      Renault
Merc 240D          62 24.4 normal power efficient car Germany         Merc
Merc 230           95 22.8 normal power efficient car Germany         Merc
Merc 280          123 19.2           normal power car Germany         Merc
```

Grammar of DataSailr Script

In this section, I will explain the grammar of DataSailr script. After reading this section, the example code above will become more interpretable.

Variables and Assignment

Variables in DataSailr script work in the same meaning as in statistics. They correspond to column names of data frames. For example, the following code divides the horse power values in hp column by the number of cylinder in cyl column. The result values are assigned to hp_per_cyl column. If hp_per_cyl column does not exist yet, new column is created. Otherwise the column values are updated.

```
datasailr::sail( mtcars, '
  hp_per_cyl = hp / cyl
')
```

In general-purpose programming language like R, variables usually mean identifiers to point to some memory or some objects. However, in statistics, data frame column names are usually called variables. DataSailr scripts follow the statistics way. Variables in scripts correspond to the columns with the same name.

Special Variables

There are variables called special variables which start and end with underscores. They are listed in Table 1. They are prepared automatically, and read-only variables.

Table 1: Special Variables

Special Variables	Description
rowname	row name
_n_	row number

Types

DataSailr scripts deal with Integer, Double and String. Integer values are automatically converted to Double values when they are calculated with Double values. Between Integer or Double and String, built-in functions need to be used for conversion, which are later explained. How R data frame columns are recognized from DataSailr is summarized in Table 2. About integer, numeric and character vectors, it is straightforward. R's boolean vectors are treated as Integer of 0 or 1, and factors are treated as String.

When you write Strings in DataSailr script, you can use a pair of single quotes or a pair of double quotes. For single-quoted strings, single quote needs to be escaped by \', and for double-quoted strings. double quote needs to be escaped by \". Additionally, when using double quotes, backslashes are interpreted as a part of escape sequences. \n, \r and \t are interpreted as newline, carriage return and tab respectively.

Table 2: How R Column Types are Treated in DataSailr

Column of R Data Frame	DataSailr
integer vector	Integer
numeric (real or decimal) vector	Double
character vector	String
boolean vector	Integer(1 for TRUE, 0 for FALSE)
factor	String

In DataSailr, there are two more types, Boolean and Regular Expression. Both of them are not used for data frame values, but are used for switching flows and string pattern matching. Boolean type values usually appear within if-else's condition parts, and are used to switch execution flows based on the Boolean type values. Regular Expressions are usually used with regular expression matching operator, which will be explained later in Regular Expression section. It is redundant to write the same regular expression repeatedly, and regular expressions are allowed to be reused by assigning to some variable, though this assignment does not affect data frame. This technique is used in the example script above.

Arithmetic Operators

The following operators in Table 3 can be applied to numbers, or Integers and Doubles. Exceptionally, only plus sign operator (+) can also be applied to strings, and concatenates them.

Table 3: Arithmetic Operators

Operator	Explanation
+	add
-	subtract
*	multiply
/	divide
**	power
^	power

If-else statement

DataSailr script only allows if-else statement as its control flow structure. There are no 'for' or 'while' loops. The result of evaluating if's condition part needs to be DataSailr's Boolean type, which is true or false. As to be mentioned, comparison operators, regular expression matching operator and logical operators return Boolean type values, and they can be used within if's condition parts.

Comparison and Logical Operators

Comparison and logical operators are listed in Table 4. Equal-to operator can be used for numbers and strings. Other comparison operators are used for numbers. Logical operators are used for Boolean values.

Table 4: Comparison and Logical Operators

Operator	Explanation
==	equal to
>	is larger than
>=	is larger than or equal to
<	is smaller than
<=	is smaller than or equal to
&&	logical operator AND
||	logical operator OR
!	unary logical operator NOT

Built-in Functions

In DataSailr script, built-in functions can be used. Currently, most of the built-in functions are for manipulating string values. Table 5 shows a list of built-in functions. These functions only conduct calculation within each row and return values, and can be assigned to variables.

Table 5: Built-in Functions

Function	Return Type	Description
num_to_str(num)	String	convert number to string
str_strip(str)	String	remove white spaces on both sides
str_lstrip(str)	String	remove white spaces on the left side
str_rstrip(str)	String	remove white spaces on the right side
str_concat(str1, str2 ...)	String	concatenate strings
str_repeat(str , num)	String	repeat str in num times
str_subset(str, num1, num2)	String	subset str from num1 to num2. Index starts from 1 (UTF8 compatible)
str_to_num(str)	Integer or Double	convert string to number
rexp_matched(num)	String	backreference for the last regular expression matched sub strings
print(str1, str2 ...)	NA	print out values (for debug purpose)

DataSailr Specific Functions

There are also other types of functions such as discard!() and push!() as listed in Table 6. These functions are used to delete rows and push additional rows respectively. discard!() function drops the current row from the result data frame. A common usage of this function is to filter out specific rows with the use of if-else statements. push!() function pushes the current variable values onto the result data frame. Even after push!() function is executed, the script execution continues on the same input row. This results in creating multiple rows from one input row. For example, it is useful to convert wide format data frame into long format.

The behaviors of these functions are different from builtin-functions in that the number of data frame rows changes. These functions are not implemented in the same way as built-in functions, but are implemented in a different layer, which is why they are called DataSailr Specific Functions. Details are mentioned in source repository's README file[5].

Table 6: DataSailr Specific Functions

Function	Description
discard!()	drop current row
push!()	push the current variable values as an additional row

Missing Values

In DataSailr scripts, single dots (.) represent missing values for numbers, and they are equivalent to NA values in data frame's numeric columns. For strings, there are not special signs to represent missing values, but empty strings ("") represent missing values. It should be noted that NA values in data frame's string columns are dealt as empty strings, and also empty strings in data frames are dealt as empty strings. When you need to differentiate NA and empty strings in data frames, you need to convert one of them to some string before applying it to DataSailr.

Table 7: Missing Value Representation

Missing value	DataSailr
numeric missing value	.
string missing value	""

The following code shows how missing values work in DataSailr script. In name column, empty strings are converted to "Anonymous", and NA is treated as an empty string. Arithmetic calculation with NA results in NA. Assigning missing value(.) to overweight column results in NA.

```
col1 = c("Tom", "Jack", NA, "Henry", "Bob")
col2 = c(1.7, 1.5 , 1.5 , 1.65 ,  1.5)
col3 = c(50 , 60  , 80  , NA   ,  100)
df = data.frame(name = col1, height = col2, bw = col3)

datasailr::sail( df, '
// missing value in name string column is set  "Anonymous"
if(name == ""){name = "Anonymous"}

// calculation with missing values results in missing values
bmi = bw / (height ^ 2)
if(bmi >= 40){ overweight = 4 }
else if( bmi >= 30 ){ overweight = 3 }
else if( bmi >= 25 ){ overweight = 2 }
else if( bmi >= 20 ){ overweight = 1 }
else { overweight = . }  // When bmi is less than 20 or missing.
')

       name height  bw      bmi overweight
1       Tom   1.70  50 17.30104         NA
2      Jack   1.50  60 26.66667          2
3 Anonymous   1.50  80 35.55556          3
4     Henry   1.65  NA       NA         NA
5       Bob   1.50 100 44.44444          4
```

[5]https://github.com/niceume/datasailr/blob/master/README.md

Regular Expression

Regular expressions can be used to check whether string values have specific patterns of character sequences, and also can be used to extract sub strings. In DataSailr script, regular expression literal is `re/pattern/` which starts with re/ and ends with `/`. Regular expression matching can be conducted using regular expression matching operator =˜. When regular expression matches strings, it results in true, otherwise false. This matching operator can be used in if-else's condition parts, and data processing can be switched based on the matching result. In the example code above, regular expressions are used for pattern matching with automobile names.

The regular expression engine[6] used in DataSailr is the same one as the default regular expression engine in Ruby[7]. DataSailr's regular expression syntax and functionalities are similar to Ruby's one. Metacharacters, escapes, character classes, repetition, capturing, alternation, character properties and anchors are supported[8]. Backreference can be used for the purpose of extracting sub strings, and rexp_matched() function extract nth sub string from the last matched string. This function is also used in the example code above.

Limitations

I have described how to use **datasailr** package, grammar of DataSailr script, and how useful it is. There are also some limitations. First, users need to learn DataSailr script which is different from R script, though it is simple and easy to learn. Second, the performance is not good compared to R's default way and **dplyr** way, partially because performance optimization is not enough conducted yet. The result of benchmarks is listed in Table 8.

```
# code to create million rows of data frame
data(mtcars)
mtcarsMillion = data.frame()
n = 30000 # Add mtcars 30000 times
mtcarsMillion = do.call("rbind", replicate(n, mtcars , simplify = FALSE))

# codes to add new column with benchmarking
# R-base
system.time({
result = mtcarsMillion
result["hp_per_cyl"] = mtcarsMillion["hp"] / mtcarsMillion["cyl"]
})

# dplyr
system.time({
result = dplyr::mutate(mtcarsMillion, hp_per_cyl = hp / cyl )
})

# datasailr
system.time({
result = datasailr::sail(mtcarsMillion, code='
  hp_per_cyl = hp / cyl
')
})

# dplyr using rowwise()
library(magrittr)
system.time({
result = dplyr::rowwise(mtcarsMillion) %>%
  dplyr::mutate( hp_per_cyl = hp / cyl )
})
```

[6]Onigmo https://github.com/k-takata/Onigmo
[7]http://www.ruby-lang.org
[8]https://docs.ruby-lang.org/en/3.0.0/doc/regexp_rdoc.html

Table 8: Performance Comparison

Method	Elapsed Time (seconds)
R-base(4.0.4)	0.159
dplyr(1.0.4)	0.471
datasailr(0.8.7)	3.171
dplyr using rowwise()	10.777

Summary

The **datasailr** package offers a yet another way to process data. Its row by row data processing approach using DataSailr script can be intuitive and can improve readability. This approach also differentiates data manipulation steps from statistical analysis and machine learning steps, which allows users to focus on each step. This package shows potential benefits using domain specific language(DSL) for data processing..

Acknowledgement

I would like to thank my family, Yuki, Joichiro and Kaoruko for always giving me joy.

Bibliography

S. M. Bache and H. Wickham. *magrittr: A Forward-Pipe Operator for R*, 2020. URL `https://CRAN.R-project.org/package=magrittr`. R package version 2.0.1. [p1]

G. Grothendieck. *sqldf: Manipulate R Data Frames Using SQL*, 2017. URL `https://CRAN.R-project.org/package=sqldf`. R package version 0.4-11. [p1]

SAS Institute. *SAS user's guide: Statistics*, volume 2. Sas Inst, 1985. [p1]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2021. URL `https://CRAN.R-project.org/package=dplyr`. R package version 1.0.4. [p1]

Toshihiro Umehara
NA
Japan
`toshi@niceume.com`