post01-data frame

Lona Dai

Introduction

We have been using data frames quite a lot in our labs or homeworks. When we want to manipulate different types of data, data frames will be helpful because unlike matrices, which can only be applied for data with the same type, data frames can create a table that contains data with different types. From our frequent choice of data.frame when we deal with various data, I will explore more about data frames in this post and discuss the use of data frames as well as the advantages of data frames.

If you are not familiar with data structures which are matrices, vectors, lists, arrays, dataframes, or even data types, I will recommend that you watch this video (ref1) first before moving on to my exploration of data frames. This video will give you the basic ideas about data types and data structures and also their relationships.

Proceed

Now, I'm sure you are ready for my exploration.

When we want to check whether the variable we see is a data frame or not, we may first come up with typeof() for checking. But the result of typeof(a data frame) will display "list" which might give us a wrong impression and lead us to consider it as a list or a matrix. In fact, "a data frame is a special case of list". (ref2) Therefore, to clearly check the identity of data frames, we can use class() function which will directly outputs "data.frame" or we can just simply use is.data.frame() function. (ref4) In the examples I came up below, I made the matrix with inputs in a list for a better comparison with data frame.

```
dataframe = data.frame("Character" = c("one", "two", "three"), "Numeric" = 1:3, "logical" = c(TRUE, FALSE, TRUE))

dataframe
```

```
##   Character Numeric logical
## 1       one       1    TRUE
## 2       two       2   FALSE
## 3     three       3    TRUE
```

```
matrice = matrix(list(1,2,3,4,5,6,7,8,9), nrow = 3, ncol = 3)
matrice
```

```
##      [,1] [,2] [,3]
## [1,] 1    4    7
## [2,] 2    5    8
## [3,] 3    6    9
```

```
typeof(dataframe)
```

```
## [1] "list"
```

```
typeof(matrice)
```

```
## [1] "list"
```

```
class(dataframe)
```

```
## [1] "data.frame"
```

```
is.data.frame(dataframe)
```

```
## [1] TRUE
```

```
class(matrice)
```

```
## [1] "matrix"
```

```
is.data.frame(matrice)
```

```
## [1] FALSE
```

From the data frame we created above, we can see under column "Character", the string inputs in that column are marked as factors. This is not the character type we wanted. The reason why this happens is data.fram() function sets stringAsFactors = TRUE in defalut and therefore, when we create a data frame without specifying stringAsFactors to be FALSE, the function will automatically convert our strings into factors. To resolve this problem, we can simply do:

```
dataframe = data.frame("Character" = c("one", "two", "three"), "Numeric" = 1:3, "logical" = c(TRUE, FALSE, TRUE),
stringsAsFactors = FALSE)
dataframe
```

```
##   Character Numeric logical
## 1       one       1    TRUE
## 2       two       2   FALSE
## 3     three       3    TRUE
```

After creating a data frame that we satisfy, we can easily get access to its content by using [ ], [[ ]], $ because its type is “list”, or we can use the methods we use for accessing matrices. ref2

```
dataframe["Numeric"]
```

```
##   Numeric
## 1       1
## 2       2
## 3       3
```

```
dataframe$Numeric
```

```
## [1] 1 2 3
```

```
dataframe[["Numeric"]]
```

```
## [1] 1 2 3
```

```
dataframe[[2]]
```

```
## [1] 1 2 3
```

```
dataframe[ , 2]
```

```
## [1] 1 2 3
```

```
dataframe[ , "Numeric"]
```

```
## [1] 1 2 3
```

```
dataframe[ , 2:3]
```

```
##   Numeric logical
## 1       1    TRUE
## 2       2   FALSE
## 3       3    TRUE
```

```
dataframe[ , c("Numeric", "logical")]
```

```
##   Numeric logical
## 1       1    TRUE
## 2       2   FALSE
## 3       3    TRUE
```

```
dataframe[2, ]
```

```
##   Character Numeric logical
## 2       two       2   FALSE
```

We can see the difference between **dataframe[“Numeric”]** and **dataframe[[“Numeric”]]**. The first one’s result is a data frame with that column name “Numeric” while the second one’s result only shows the inputs/data under that column name. And the difference between **dataframe[,2]**, **dataframe[, 2:3]** and **dataframe[2,]** is when accessing one specific column, the column name will not output, and accessing two or more columns or accessing rows, the column names will be shown.

We can also add columns or rows like what we did in matrix using cbind(), rbind(). But since **dataframe** is a data frame, we need to combine another data frame with it in order to add columns and rows. (ref4) Keep in mind that the number of inputs in that column and the names of columns must match with the original data.

```
cbind(dataframe, data.frame("Add" = 2:4))
```

```
##   Character Numeric logical Add
## 1       one       1    TRUE   2
## 2       two       2   FALSE   3
## 3     three       3    TRUE   4
```

```
rbind(dataframe, data.frame("Character" = "four", "Numeric" = 4, "logical" = FALSE))
```

```
##   Character Numeric logical
## 1       one       1    TRUE
## 2       two       2   FALSE
## 3     three       3    TRUE
## 4      four       4   FALSE
```

Addition notes: If you feel uncomfortable with matrix accessing methods, let's refresh our memory of accessing matrices.(ref3) Since the variable **matrice** has inputs as list, the outputs may not seem obvious. I will simply change the inputs to integers for a better view in markdown. Even if the matrix has assinged names for rows and colums, it will not affect the output. This is why I used two different ways (column number and column name) in the above example to access the second and third colums and the outputs are the same.

```
matrice = matrix(1:9, nrow = 3, ncol = 3)
matrice
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
matrice[2, 1]
```

```
## [1] 2
```

```
matrice[, 2]
```

```
## [1] 4 5 6
```

```
matrice[2, ]
```

```
## [1] 2 5 8
```

After knowing how to create and access data frames, we can now consider the advantages of using data frames. The first advantage is obviously the use of different types of data. The second advantage is data frame "shares properties of both the matrix and the list" and this is why we can access the data in the data frame by using both list accessing methods and matrix accessing methods. (ref4). The third advantage is that the data frame can convert itself into a matrix but as a result, the different types of data will become the same type. (ref5, ref6) Let's see some examples:

```
data.matrix(dataframe)
```

```
## Warning in data.matrix(dataframe): NAs introduced by coercion
```

```
##      Character Numeric logical
## [1,]        NA       1       1
## [2,]        NA       2       0
## [3,]        NA       3       1
```

```
as.matrix(dataframe)
```

```
##      Character Numeric logical
## [1,] "one"     "1"     " TRUE"
## [2,] "two"     "2"     "FALSE"
## [3,] "three"   "3"     " TRUE"
```

We can see from the examples, **data.matrix()** converts the data frame into numberic matrix which by description on the website of ref4, it returns a matrix "by converting all the variables in a data frame to numeric mode and then binding them together as the columns of a matrix. Factors and ordered factors are replaced by their internal codes." And **as.matrix()** converts the data frame into a matrix with data type character. This is because in ref5, it says "if you convert a dataframe with different classes of columns, then your matrix will just be all character".

The fourth advantage is that data frame allows you to treat the inputs as factors so that you can use a lot more functions to help you explore and analyse the table of data. (ref7)

```
str(dataframe)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ Character: chr  "one" "two" "three"
##  $ Numeric  : int  1 2 3
##  $ logical  : logi  TRUE FALSE TRUE
```

```
head(dataframe)
```

```
##   Character Numeric logical
## 1       one       1    TRUE
## 2       two       2   FALSE
## 3     three       3    TRUE
```

Take Home Message

Data frame is the combination of Matrix and List, thus data frames can make use of the methods for matrices and lists for accessing data and can include data with different types. It’s important that the number of rows and names of columns match with the original data frame when adding columns or rows. A data frame can also be converted into numeric matrix or a matrix with “character” data type.