Post2-Daniel-Kang

Daniel Kang

11/27/2017

install and load the package is the most important thing!

```
#Do not remember to library() it!!
library(reshape2)
library(readr)
post2 <- read.csv("./nba_2017_real_plus_minus.csv")# this is the url to get the data, https://www.kaggle.com/noahgift/social-power-nba/data, if you want, you can download the data and use `readr` to read it in csv form.
post2 <- head(post2, 5) #To cut the data so it will not look that messy :0
```

Introduction:

When you want to organize a data, you will definitely think about `dplyr`. But besides `dplyr`, I found a really interesting package online called `reshape2`. `reshap2` and `dplyr` are both made by the legend in R-community – `Hadley Wickham`.

I like this package is because this package has its own "process" to dealt with data, no matter what kind of data such as array, list or data frame, all it does is, melt the data and then cast it to varies data, it is just like melt a chocolate and cast it to different kinds or candy! It is so cool, isn't it!!!!!!!

Although I spend a lot and was also confusing in the beginning when I was learning the `melt` and `cast` functions, once you realize how to use it, OMG everything just turns out so much easier!!!

Let's Go Check It Out!!

Simple Example

This is our original data

```
#set up a random data
a <- array(c(1:23, NA), c(2,3,4))
a
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
##
## , , 4
##
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   NA
```

I want to melt to let it be organize.

```
#And now we can finally start to `melt` it!!
a <- array(c(1:23, NA), c(2,3,4))
b <- melt(a,varnames = c(1,2,3) ,value.name = "value") #using melt function
head(b,5) #head function for organizing data
```

```
##   1 2 3 value
## 1 1 1 1     1
## 2 2 1 1     2
## 3 1 2 1     3
## 4 2 2 1     4
## 5 1 3 1     5
```

And we can see there is a "NA" in the data so we can also use na.rm function to remove it.

```
melt(a,varnames = c(1,2,3) ,value.name = "x",na.rm = TRUE)
```

```
##    1 2 3  x
## 1  1 1 1  1
## 2  2 1 1  2
## 3  1 2 1  3
## 4  2 2 1  4
## 5  1 3 1  5
## 6  2 3 1  6
## 7  1 1 2  7
## 8  2 1 2  8
## 9  1 2 2  9
## 10 2 2 2 10
## 11 1 3 2 11
## 12 2 3 2 12
## 13 1 1 3 13
## 14 2 1 3 14
## 15 1 2 3 15
## 16 2 2 3 16
## 17 1 3 3 17
## 18 2 3 3 18
## 19 1 1 4 19
## 20 2 1 4 20
## 21 1 2 4 21
## 22 2 2 4 22
## 23 1 3 4 23
```

The motivation:

During this semester, I learned a lot of functions in R and everything function have their wheelhouse. But every time when I am dealing with some data or do something in R, I always get some errors such that, “the length is not same”, “ vector can not compare with array”. I am always confused and also feel annoyed because when we trying to make a statistic for something, sometimes we need to combine a lot of data, and put it into the same data frame.

So I found reshape on some website and I realize this is the package that I need, It will just make my life easier. I can combine two entirely different data together! Let’s learn from the BASE.

The history of `reshape2`

R default functions are already include reshape but to make it a better version, Hadley Wickham created the `reshape2`. Let’s just list up some changes between `reshape1` and `reshap2`. his version improves speed at the cost of functionality, so I have renamed it to reshape2 to avoid causing problems for existing users. Based on user feedback I may reintroduce some of these features.

What’s new in `reshape2`:

1, considerably faster and more memory efficient thanks to a much better underlying algorithm that uses the power and speed of subsetting to the fullest extent, in most cases only making a single copy of the data.

2, cast is replaced by two functions depending on the output type: `dcast` produces data frames, and `acast` produces matrices/arrays.

3, multidimensional margins are now possible: `grand_row` and `grand_col` have been dropped: now the name of the margin refers to the variable that has its value set to (all).

4, some features have been removed such as the | cast operator, and the ability to return multiple values from an aggregation function. I’m reasonably sure both these operations are better performed by `plyr`.

5, A new cast syntax which allows you to reshape based on functions of variables (based on the same underlying syntax as plyr):

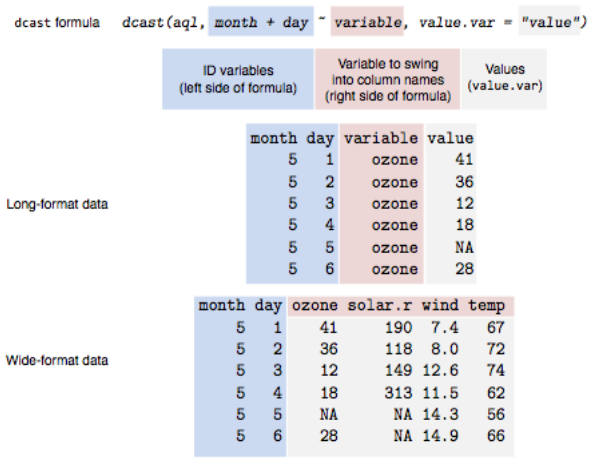
6, better development practices like namespaces and tests.

7, the function melt now names the columns of its returned data frame Var1, Var2, …, VarN instead of X1, X2, …, XN.

8, the argument `variable.name` of `melt` replaces the old argument variable_name.

P.s. Initial benchmarking has shown `melt` to be up to 10x faster, pure reshaping `cast` up to 100x faster, and aggregating `cast()` up to 10x faster.

Let me show you why `reshape2` is so interesting and how easy and convient is it. This is kinda the things I want to show you today by using the same data. The main idea is making the data, so in the future, when you tryna make the data better, you can use `reshape2` to deal with it in a minute.



Now I want to show you guys how to use `reshape2` function.

Melt function

```
####we have to melt the data first
NAME <- post2$NAME
MPG <- post2$MPG
DRPM <- post2$DRPM
ORPM <- post2$ORPM
RPM <- post2$RPM
WINS <- post2$WINS
GP <- post2$GP
md <- head(melt(post2, id=c('TEAM','MPG'))) ##This is how original data frame look like
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
md
```

```
##   TEAM  MPG variable                value
## 1  CLE 37.8     NAME      LeBron James, SF
## 2   GS 33.4     NAME     Stephen Curry, PG
## 3  CHI 37.0     NAME      Jimmy Butler, SG
## 4  OKC 34.6     NAME Russell Westbrook, PG
## 5   GS 32.5     NAME    Draymond Green, PF
## 6  CLE 37.8       GP                    74
```

Cast function

The first three functions are very easy. We just pick up the variable and also assign what data are we gonna use.

ow that you have a molten data you can reshape it into a data frame using dcast function or into a vector/matrix/array using the acast function. The basic arguments of *cast is the molten data and a formula of the form x1 + x2 ~ y1 + y2. The order of the variables matter, the first varies slowest, and the last fastest. There are a couple of special variables: "…" represents all other variables not used in the formula and "." represents no variable, so you can do formula = x1 ~ .

It is easier to understand the way it works by doing it yourself. Try different options and see what happens:

```
dcast(md, WINS+MPG~variable)
```

```
## Warning in cbind(MPG = structure(c(5L, 2L, 4L, 3L, 1L, 5L), n = 5L), WINS
## = structure(c(5L, : number of rows of result is not a multiple of vector
## length (arg 2)
```

```
##    WINS  MPG                 NAME   GP
## 1 16.84 32.5    Draymond Green, PF <NA>
## 2 17.34 34.6 Russell Westbrook, PG <NA>
## 3 17.35 37.0      Jimmy Butler, SG <NA>
## 4 18.80 33.4     Stephen Curry, PG <NA>
## 5 20.43 37.8      LeBron James, SF   74
```

```
dcast(md, WINS+variable~MPG)
```

```
## Warning in cbind(variable = structure(c(1L, 1L, 1L, 1L, 1L, 2L), n = 2L), :
## number of rows of result is not a multiple of vector length (arg 2)
```

```
##    WINS variable               32.5              33.4
## 1 16.84     NAME Draymond Green, PF              <NA>
## 2 17.34     NAME               <NA>              <NA>
## 3 17.35     NAME               <NA>              <NA>
## 4 18.80     NAME               <NA> Stephen Curry, PG
## 5 20.43     NAME               <NA>              <NA>
## 6    NA       GP               <NA>              <NA>
##                    34.6               37             37.8
## 1                  <NA>             <NA>             <NA>
## 2 Russell Westbrook, PG             <NA>             <NA>
## 3                  <NA> Jimmy Butler, SG             <NA>
## 4                  <NA>             <NA>             <NA>
## 5                  <NA>             <NA> LeBron James, SF
## 6                  <NA>             <NA>               74
```

```
dcast(md, WINS~variable+MPG)
```

```
## Warning in cbind(structure(c(5L, 4L, 3L, 2L, 1L), n = 5L),
## structure(c(5L, : number of rows of result is not a multiple of vector
## length (arg 1)
```

```
##    WINS          NAME_32.5         NAME_33.4             NAME_34.6
## 1 16.84 Draymond Green, PF              <NA>                  <NA>
## 2 17.34               <NA>              <NA> Russell Westbrook, PG
## 3 17.35               <NA>              <NA>                  <NA>
## 4 18.80               <NA> Stephen Curry, PG                  <NA>
## 5 20.43               <NA>              <NA>                  <NA>
##            NAME_37        NAME_37.8 GP_37.8
## 1             <NA>             <NA>    <NA>
## 2             <NA>             <NA>    <NA>
## 3 Jimmy Butler, SG             <NA>    <NA>
## 4             <NA>             <NA>    <NA>
## 5             <NA> LeBron James, SF      74
```

```
#Can you see the differences between these three funcitons?

#Yes!! You can pick up the variable you would like to set as a column or row, it is so easy!!
```

This is what the data should look like after we melt it.

```
#This is what the data should look like after we melt it.
head(melt(post2, id = c("MPG", "RPM"), variable.name = "type", value.name = "val"))
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
##    MPG  RPM type                  val
## 1 37.8 8.42 NAME      LeBron James, SF
## 2 33.4 7.41 NAME     Stephen Curry, PG
## 3 37.0 6.62 NAME      Jimmy Butler, SG
## 4 34.6 6.27 NAME Russell Westbrook, PG
## 5 32.5 7.14 NAME    Draymond Green, PF
## 6 37.8 8.42 TEAM                  CLE
```

Using `str` to check how does the data look like after melted it.

```
str(melt(post2, id = c("MPG", "RPM"), variable.name = "type", value.name = "val"))
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
## 'data.frame':    30 obs. of  4 variables:
##  $ MPG : num  37.8 33.4 37 34.6 32.5 37.8 33.4 37 34.6 32.5 ...
##  $ RPM : num  8.42 7.41 6.62 6.27 7.14 8.42 7.41 6.62 6.27 7.14 ...
##  $ type: Factor w/ 6 levels "NAME","TEAM",..: 1 1 1 1 1 2 2 2 2 2 ...
##  $ val : chr  "LeBron James, SF" "Stephen Curry, PG" "Jimmy Butler, SG" "Russell Westbrook, PG" ...
```

We can also select multiple variables

```
head(melt(post2, id = c("MPG", "RPM"), measure.vars = c("NAME", "GP"), variable.name = "type", value.name = "val")
)
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
##    MPG  RPM type                  val
## 1 37.8 8.42 NAME      LeBron James, SF
## 2 33.4 7.41 NAME     Stephen Curry, PG
## 3 37.0 6.62 NAME      Jimmy Butler, SG
## 4 34.6 6.27 NAME Russell Westbrook, PG
## 5 32.5 7.14 NAME    Draymond Green, PF
## 6 37.8 8.42   GP                    74
```

Assign `Margins`

To add statistics to the margins of your table, we can set the argument margins appropriately. Set margins = TRUE to display all margins or list individual variables in a character vector. Note that changing the order and position of the variables in the *cast formula affects the margins that can be computed.

```
head(melt(post2, id = c("MPG", "RPM"), margins = "RPM", measure.vars = c("NAME", "GP"), variable.name = "type", va
lue.name = "val"))
```

```
## Warning: attributes are not identical across measure variables; they will
## be dropped
```

```
##    MPG  RPM type                  val
## 1 37.8 8.42 NAME      LeBron James, SF
## 2 33.4 7.41 NAME     Stephen Curry, PG
## 3 37.0 6.62 NAME      Jimmy Butler, SG
## 4 34.6 6.27 NAME Russell Westbrook, PG
## 5 32.5 7.14 NAME    Draymond Green, PF
## 6 37.8 8.42   GP                    74
```

A nice peice of advice that I found online.

As mentioned above, the order of variables in the formula matters, where variables specified first vary slowest than those specified last. Because comparisons are made most easily between adjacent cells, the variable you are most interested in should be specified last, and the early variables should be thought of as conditioning variables.

Take-home message:

In this post, I introduce a little bit about reshape2, I found this online when I need to reshape my data frame when I was doing the homework. It is really helpful, easy and fast. All you need to do is just melt the data and cast it! I have some example above and good luck for final. Instead introduce some FUN package or stuff, I will more like to talk about some useful package and can helps us in the future. transform the data have a lot of ways and functions but reshape2 is definitely one of the best package and the best choice for beginner! Hope you have a good day and good luck for final.

References

1. reshape2
2. reshape2
3. reshape2
4. reshape2
5. reshape2
6. reshape2
7. reshape2
8. reshape2
9. reshape2