Post 2

Tianqi Lu

11/29/2017

Introduction to “reshape2” Package

Introduction

This post is to introduce a very useful package named “reshape2” in R, and also we will learn some important and useful functions in it!

Motivation

Last time, we learned a package named “tidyr”, which is a good package we cna use to “tidy” data so that the data is easier to understand and manipulate with. But when I google related stuff, I found that there was a package even more popular than “tidyr” with similar functionality, thus I want to dig into this package “reshape2” this time and do some comparison between the two packages.

Background

> The main purpose of the package “reshape2” is to, as we can guess from the name, “RESHAPE” the data, so that we can more easily work on data manipulation, visualization and modalization process.
> Here is a brief list of the functions in this package:
> melt_check, french_fries, guess_value, melt.data.frame, cast, melt, recast, colsplit, dcast etc.

> Here is the basic background of this package. Check it out if want more information.

Key Points

There are two major goals for the package “reshape2”, one is to reshape data, the other is to aggregate.
The first major goal is still to reshape data so that it is easier to deal with, but with a different approach this time. Just a reminder from last time that “tidy” data is defined as follows:

- Each column is a variable.
- Each row is an observation.
- Each value is a cell.

This way of interpretating data makes users easier to work with it. We can always refer to variables as column names and refer to observations as row indices.
As for the second goal, we will introduce in the second half.

Major Functions

The two mostly used functions in this package are “melt()”, “colsplit()” and “dcast()”

- melt(): you “melt” data so that each row is a unique id-variable combination: it makes “wide” data long.
- colsplit(): split columns.
- dcast(): takes long-format data and casts it into wide-format data。

Examples

First, we need to download and import this package.

```
install.packages("reshape2")
```

Then, we load this package

```
library(tidyr)
library(reshape2)
library(dplyr)
```

We can get started with the example now!

melt() function

First, let’s see an original table

```
original = data.frame(subject = c("John", "Mary", "Mike", "Lily"),
                      age = c(33,30,27,21),
                      weight = c(90, 75, 96, 80),
                      height = c(185, 170, 176, 168))
original
```

```
##   subject age weight height
## 1    John  33     90    185
## 2    Mary  30     75    170
## 3    Mike  27     96    176
## 4    Lily  21     80    168
```

This is table now is of the form data frame. However, sometimes, what we want is not a data frame but, what we call, pairs of identifiers and measured variables. In this specific case, “id” is subject, or name, and all other columns are measured variables. Therefore, what we want to do sometimes is to have only one measured variable in a row which makes the chart easier to look. To solve this problem, we could either use “gather()” function in “tidyr” package or “melt()” function in “reshape2” package. Let’s have a look at it!

```
## Use melt function in "reshape2"
moltenTable <- melt(original, id = 'subject')
moltenTable
```

```
##    subject variable value
## 1     John      age    33
## 2     Mary      age    30
## 3     Mike      age    27
## 4     Lily      age    21
## 5     John   weight    90
## 6     Mary   weight    75
## 7     Mike   weight    96
## 8     Lily   weight    80
## 9     John   height   185
## 10    Mary   height   170
## 11    Mike   height   176
## 12    Lily   height   168
```

```
# Use gather function in "tidyr"
gatheredTable <- gather(original, "subject", "value")
gatheredTable
```

```
##    subject subject value
## 1     John     age    33
## 2     Mary     age    30
## 3     Mike     age    27
## 4     Lily     age    21
## 5     John  weight    90
## 6     Mary  weight    75
## 7     Mike  weight    96
## 8     Lily  weight    80
## 9     John  height   185
## 10    Mary  height   170
## 11    Mike  height   176
## 12    Lily  height   168
```

Note that, we got two tables with same entries in it. In this form, each row has only one observation which makes more sense in some cases. However, sometimes, we require the reverse step of it, and we want a data frame that is to compute with. That is where we will use “dcast()” function in “reshape2” package.

“dcast()” function

Sometimes you may encounter a data frame as above, and you may think, oh god, this is so hard to use because you are used to “$” notation. Then what should we do? To solve such a problem, we use “dcast()” function to make long data wider.
(Note: There are many cast() functions, but since under most circumstances, we want to use a data frame to compute data, we use dcast() function, aka “data frame cast() function”.)

```
#dcastTable <- dcast(moltenTable)
# Use "spread()" function in "tidyr"
spreadTable <- spread(moltenTable, key = "variable", value = "value")
spreadTable
```

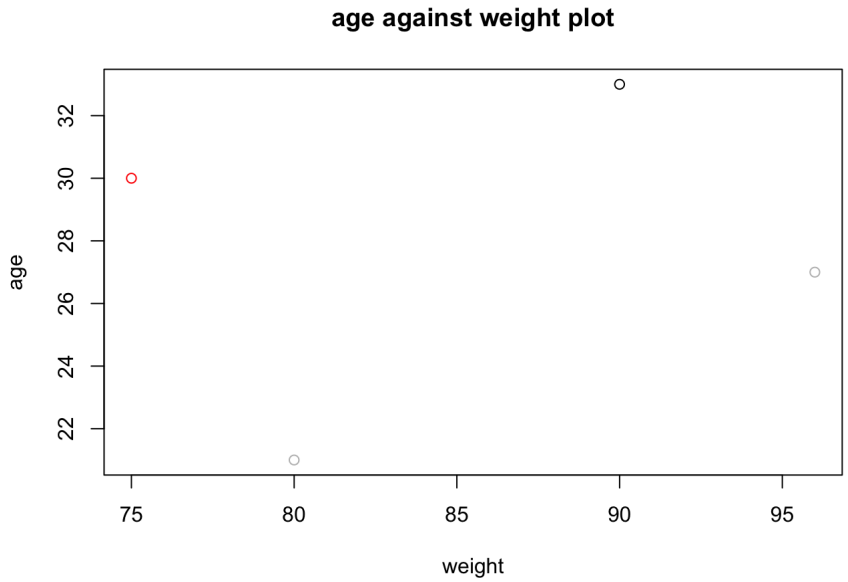
```
##   subject age weight height
## 1    John  33     90    185
## 2    Lily  21     80    168
## 3    Mary  30     75    170
## 4    Mike  27     96    176
```

```
# Use "dcast()" function in "reshape2"
dcastTable <- dcast(moltenTable, formula = subject ~ variable, value.var = "value")
dcastTable
```

```
##   subject age weight height
## 1    John  33     90    185
## 2    Lily  21     80    168
## 3    Mary  30     75    170
## 4    Mike  27     96    176
```

We got two tables with same entries again! This new generated table is more friendly when we want to use functions such as “filter()” in “dplyr” package. This is what we call a “tidy” table.
Now suppose you want to plot the age versus weight table, we can easily do it!

```
plot(dcastTable$age ~ dcastTable$weight, main = "age against weight plot", xlab = "weight", ylab = "age", col = dc
astTable$height)
```



Alright, now you are gonna ask, if these two packages have such similar major functions, why would the second one be created? I would say that’s a terrific question, and this is what we will introduce in the following section.

Aggregation using “dcast()” function

```
# To illustrate how to aggregate using dcast() function, we use the built in data frame "tips" in "reshape2" packa
ge
head(tips)
```

```
##   total_bill  tip    sex smoker day   time size
## 1      16.99 1.01 Female     No Sun Dinner    2
## 2      10.34 1.66   Male     No Sun Dinner    3
## 3      21.01 3.50   Male     No Sun Dinner    3
## 4      23.68 3.31   Male     No Sun Dinner    2
## 5      24.59 3.61 Female     No Sun Dinner    4
## 6      25.29 4.71   Male     No Sun Dinner    4
```

Suppose now our task is to calculate, for this specific restaurant, when the most tip percentage occurs.

```
# First we can use melt() function to make the data frame easier to look
molten <- melt(tips, id = c("sex", "smoker", "day", "time", "size"))

# We can use dcast() function to get the aggregated data
aggregatedTable <- dcast(molten, day + time ~ variable, mean)

# Then we add new column to it
aggregatedTable$percentage <- aggregatedTable$tip / aggregatedTable$total_bill
aggregatedTable
```

```
##    day   time total_bill      tip percentage
## 1  Fri Dinner   19.66333 2.940000  0.1495169
## 2  Fri  Lunch   12.84571 2.382857  0.1854982
## 3  Sat Dinner   20.44138 2.993103  0.1464238
## 4  Sun Dinner   21.41000 3.255132  0.1520379
## 5 Thur Dinner   18.78000 3.000000  0.1597444
## 6 Thur  Lunch   17.66475 2.767705  0.1566795
```

```
# Last, we find the maximum
aggregatedTable[which.max(aggregatedTable$percentage),]
```

```
##   day  time total_bill      tip percentage
## 2 Fri Lunch   12.84571 2.382857  0.1854982
```

Therefore, we could easily draw the conclusion that customers tend to give most tip for Friday lunch. Note that, we may be able to get the same value using traditional aggregate() function, but aggregate() has its own strenth, and for this specific question, using aggregate() looks messier than this solution.

“colsplit()” function

This part we will introduce the last comparison between the two packages, between “colsplit()” function in “reshape2” package and “separate()” function in “tidyr” package.
“colsplit()” function splits A vector into multiple columns. It is useful for splitting variable names that a combination of multiple variables. Uses type.convert to convert each column to correct type, but will not convert character to factor.

```
# Suppose this is a table we want to split
x <- data.frame(
  YO = c("a_1", "a_2", "b_1", "c_3")
)
x
```

```
##    YO
## 1 a_1
## 2 a_2
## 3 b_1
## 4 c_3
```

Let’s say, we want to split this into two columns, one of which containing the letters, while the other containing the numbers. How should we approach? Last time we introduced “separate()” function in “tidyr” package to solve this problem, but this time we will do something similar with function “colsplit()” in “reshpae2” package.

```
# Use separate() function
x %>% separate(YO, c("Letter", "Number"))
```

```
##   Letter Number
## 1      a      1
## 2      a      2
## 3      b      1
## 4      c      3
```

```
# Use colsplit() function
colsplit(x$YO, pattern = "_", names = c("Letter", "Number"))
```

```
##   Letter Number
## 1      a      1
## 2      a      2
## 3      b      1
## 4      c      3
```

Great! We got same results again1 This shows that the colsplit() funciton is just as useful as separate() function!
These are basically the most common used functions in this package. If you feel unsatisfied and want to get more out of it, I highly recommend you to check out this video to get a more direct feeling about these functions and understand them better!

---

extracurricular ploting that helps analysing data. (useful plot involved in time series analysis.)

background

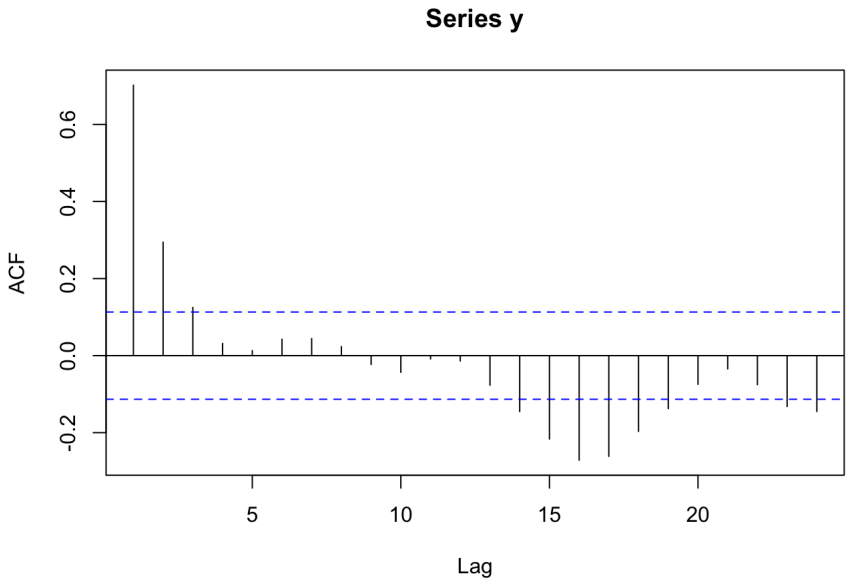
time series package
We are gonna learn something about time series in this part. This is important because many projects have stuff related to time difference, where we will apply time series. Thus we want to try some useful plots to help us interprete data.

ARMA(p,q) model: This model is the combination of two ways we relate current data point to previous values and errors, namely AR(p) (autoregressive model) and MA(q)(moving average model)

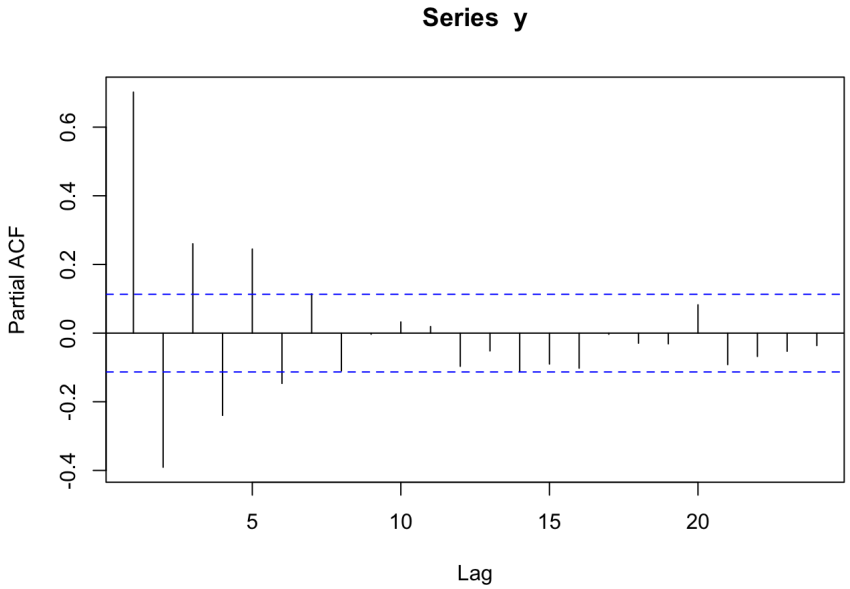
In time series we always care about the correlation between different time segment. And in time series case, we call the correlation between different time segment autocorrelation. And later when we want to analyze residuals, we are hoping these residuals can be distributed normally. Therefore we also have some really good plots that can help us assess the normality of variables.

Dealing with my data, I’m using auto regressive moving average model, which the characteristic of this model assumes that current data point has autocorrelation with value of previous time points and has correlation with error of previous time points. Therefore, we need to look and the graph of the auto correlation function that can help us to determine which time lag is significant.

```
## use acf to find how many time lag our Yt depends on error term , the correlation with timelag 1 is much more si
gnificant than other, so we may conclude this is MA(1)
acf(y)
```




```
## use pacf to find how many time lag our Yt depends on previous Yt-k, the correlation with timelag 1 is much more significant than other, so we may conclude this is AR(1)
pacf(y)
```



For Reproducibility Purposes

1. You need to download packages: “reshape2”, “tidyr” (for comparison purposes), “dplyr”(Not really necessary, there ar ealternatives), “tseries”, “TSA”, “leaps”, “mgcv”, “locfit”
2. The RStudio version is 1.0.153
3. The R version is 3.4.3
4. Commands and Functions:
    - Use library() to load the packages above.
    - Use melt(), colsplit(), dcast() to finish the other steps. (There is a step you may need to use plot() function)
    - Use acf() and pacf() to check if there is any correlation between different times
5. Data: For the first part, the data I’m using are all randomly made and you don’t need to import any data. You can either make your own data, or directly copy the steps I did above! While for the second part, you will need to import the above packages and use the built-in file ts1.RData.

Take-home Message

When we are given some “untidy” data, we can always try using “reshape2” or “tidyr” package to “tidy” the data and makes it easy to be manipulated and visualized. These two packages share a lot of similar functions for the purpose of tidying data:
Use “melt()” and “gather()” to collect multiple columns into key-value pairs, so that we can analyze the relationship between IDs and variables (keys and values). Use “dcast()” and “spread()” to take two columns (key & value) and spread into multiple columns to make data manipulation easier.
Use “colsplit()” and “separate()” if you want to split one column into two or the other way around.
But for the package “reshape2”, dcast() function provides a more useful functionality which is “aggregate”. We can achieve similar results as

aggreagte() function can do, but easier in some cases. Use “acf()” and “pacf()” function to check if there is any correlation between different times.

References

1. CRAN - Package reshape2, cran.r-project.org/web/packages/reshape2/index.html.
2. “How to reshape data in R: tidyr vs reshape2.” MilanoR, 20 June 2016, www.milanor.net/blog/reshape-data-r-tidyr-vs-reshape2/.
3. Thiagogm. “Reshape and aggregate data with the R package reshape2.” Thiago G. Martins, 31 Oct. 2013, tgmstat.wordpress.com/2013/10/31/reshape-and-aggregate-data-with-the-r-package-reshape2/.
4. Introduction to reshape2, xiangxing98.github.io/R_Learning/R_an_introduction_to_reshape2.nb.html
5. “Reshape and aggregate data with the R package reshape2.” R-Bloggers, 31 Oct. 2013, www.r-bloggers.com/reshape-and-aggregate-data-with-the-r-package-reshape2/.
6. “reshape2.” Function | R Documentation, www.rdocumentation.org/packages/reshape2/versions/1.4.2/topics/colsplit.
7. nubinshred. “Reshaping Data Using R.” YouTube, YouTube, 3 Mar. 2015, www.youtube.com/watch?v=Z8nNkDD5xjc.
8. “Data Tidying.” Data Tidying · Data Science with R, garrettgman.github.io/tidying/.
9. “Separate one column into multiple columns.” Separate one column into multiple columns. - separate • tidyr, tidyr.tidyverse.org/reference/separate.html.