Post 1

Tianqi Lu

10/29/2017

Introduction to “tidyr” Package

Introduction

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

Motivation

By now, we have learned that “dplyr” is a great package to use for data manipulation and “ggplot2” is a great package to use for data visilization. However, I noticed that, so far, the data we work on is mainly given by the professor, and these data are mostly well organized. But we will at last deal with data not given by the professor. Therefore, very naturally, a problem comes into my mind: what if the data we get in the future is no longer well organized and what should we do to proceed and get everything work smoothly? This question motivated me to google and found this package, “tidyr”.

Background

> The main purpose of this package is to, as we can guess from the name, “TIDY” 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:
> unnest, separate, extract, nest, fill, complete, pipe, unite, expand, gather, spread etc.

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

Key Points

There are three main properties of what we call “tidy” data.

- 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.

Major Functions

The two mostly used functions in this package are “gather()” and “spread()”

- gather(): takes multiple columns and gathers them into key-value pairs: it makes “wide” data longer.
- spread(): takes two columns (key & value) and spreads into multiple columns: it makes “long” data wider.

And as an extra, we will introduce two other functions: “separate()” and “unite()”.

Examples

First, we need to download and import this package.

```
install.packages("tidyr")
```

Then, we load this package

```
library(tidyr)
library(dplyr)
library(ggplot2)
```

We can get started with the example now!

spread() function

First, let’s see a “untidy” table

```
table1 <- data.frame(
  country = c(rep("Afghanistan", 4), rep("Brazil", 4), rep("China", 4)),
  year = c(1999, 1999, 2000, 2000, 1999, 1999, 2000, 2000, 1999, 1999, 2000, 2000),
  classification = c(rep(c("cases", "population"), 6)),
  value = c(745, 19987071, 2666, 20595360, 37737, 172006362, 80488, 174504898, 212258, 1272915272, 213766, 1280428
583)
)
table1
```

```
##        country year classification      value
## 1  Afghanistan 1999          cases        745
## 2  Afghanistan 1999     population   19987071
## 3  Afghanistan 2000          cases       2666
## 4  Afghanistan 2000     population   20595360
## 5       Brazil 1999          cases      37737
## 6       Brazil 1999     population  172006362
## 7       Brazil 2000          cases      80488
## 8       Brazil 2000     population  174504898
## 9        China 1999          cases     212258
## 10       China 1999     population 1272915272
## 11       China 2000          cases     213766
## 12       China 2000     population 1280428583
```

Notice that this table is not very easy to use, because the main variables here should be “cases” and “population”, however, they are not considered as columns. Consider a question: “Find the country whose population is greater than 1000000000 in year 1999.” You will find it very hard to move forward using this table. **This is where we would like to use “spread()”**. As we said earlier, “spread()” takes in “key & value”. Here, key is “classification” and value is “value”.

```
# Use spread() to get a fancier table
spreadTable <- spread(table1, key = classification, value = value)
spreadTable
```

```
##       country year  cases population
## 1 Afghanistan 1999    745   19987071
## 2 Afghanistan 2000   2666   20595360
## 3      Brazil 1999  37737  172006362
## 4      Brazil 2000  80488  174504898
## 5       China 1999 212258 1272915272
## 6       China 2000 213766 1280428583
```

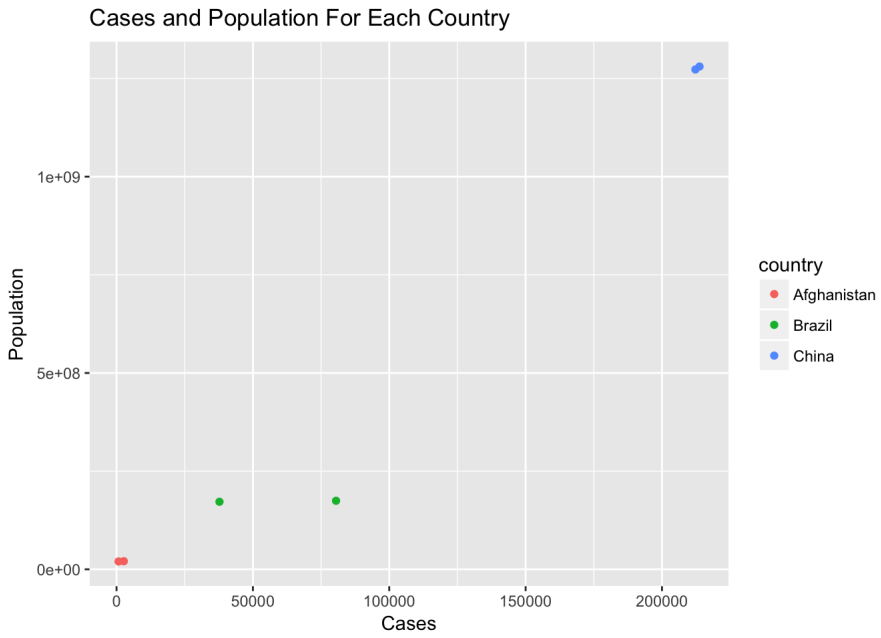
Note that this spreadTable is much easier to work with! We can easily use “dplyr” package’s function such as filter():

```
# Get the country whose population is greater than 1000000000 in year 1999
filter(spreadTable, year == 1999 & population > 1000000000)
```

```
##   country year  cases population
## 1   China 1999 212258 1272915272
```

More importantly, this spreadTable allows us to visualize data! Let’s try it out!

```
ggplot(spreadTable, aes(x = spreadTable$cases, y = spreadTable$population, col = country)) + geom_point()+ xlab("Cases") + ylab("Population") + ggtitle("Cases and Population For Each Country")
```



This plot visualizes three countries’ population and the number of cases.

Now, let’s explore some “gather()”!

gather() function

Sometimes, instead of table1, we might encounter some other messy and untidy tables. Let’s look at the following table:

```
table2 <- data.frame(
  country = c("Afghanistan", "Brazil", "China"),
  '1999' = c(745, 37737, 212258),
  '2000' = c(2666, 80488, 213766),
  check.names = FALSE
)
table2
```

```
##       country   1999   2000
## 1 Afghanistan    745   2666
## 2      Brazil  37737  80488
## 3       China 212258 213766
```

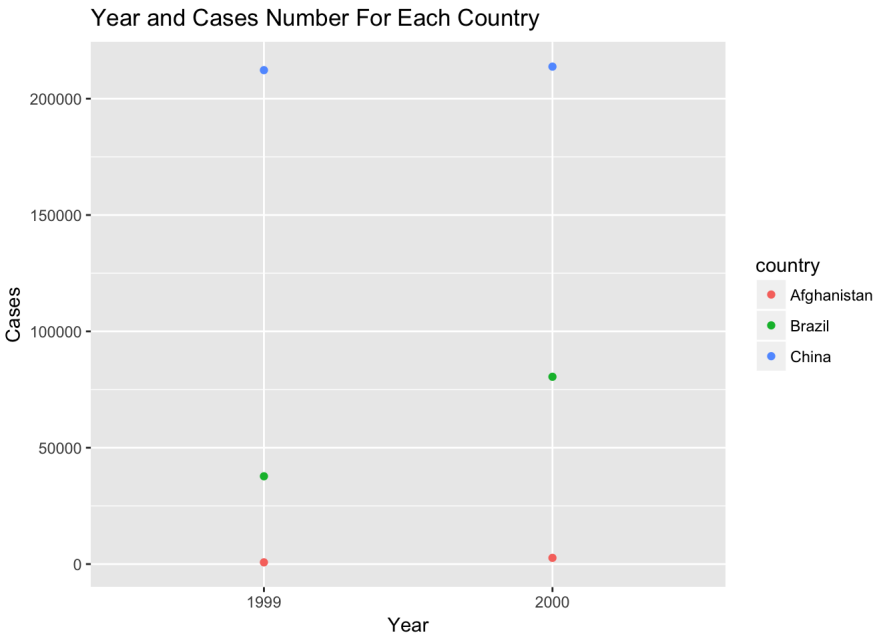
As you see in this table, the values in column 2 and 3 are actually “cases” variable values, but they are not specified as “cases”, which makes users very hard to understand what the values are or extract values. And the most important thing is that this table is a disaster when it comes to data visualization. Therefore, we need magic on it! **This is where we would like to use “gather()”!**

```
gatherTable <- gather(table2, "year", "cases", 2:3)
gatherTable
```

```
##       country year  cases
## 1 Afghanistan 1999    745
## 2      Brazil 1999  37737
## 3       China 1999 212258
## 4 Afghanistan 2000   2666
## 5      Brazil 2000  80488
## 6       China 2000 213766
```

gather() takes multiple columns, in this case, column 2 and 3. The second variable “year”corresponds to the name for previous column 2 and 3’s names, and the third variable “cases” corresponds to the values in column 2 and 3. Now, the table is fancy, easy to understand and work with. AND, we can now visualize our data!

```
ggplot(gatherTable, aes(x = gatherTable$year, y = gatherTable$cases, col = country)) + geom_point() + xlab("Year")
+ ylab("Cases") + ggtitle("Year and Cases Number For Each Country")
```



This plot visualizes three countries’ number of cases in each year.

Guess what guys?! There are more useful tools in the package! Let’s explore some other stuff!

seperate() function

Given either regular expression or a vector of character positions, separate() turns a single character column into multiple columns. This could be useful when we want to split the data in some column and analyze the relationship between them. Let’s see an example how this works!

```
# First we create a data table with one column only, containing vectors of length 2.
table3 <- data.frame(x = c(NA, "a.b", "a.d", "b.c"))
table3
```

```
##      x
## 1 <NA>
## 2  a.b
## 3  a.d
## 4  b.c
```

Now consider if we want to split this column into two columns so that we can manipulate them more easily, we'll try this!

```
table3 %>% separate(x, c("A", "B"))
```

```
##      A    B
## 1 <NA> <NA>
## 2    a    b
## 3    a    d
## 4    b    c
```

As you wish, we get two columns now!!! And you can do whatever you want to compute with these two columns.
Last but not least, there is one more useful tool, which is unite()!

unite() function

This function is basically the complement of separate() function! Let's directly see an example and get some feeling about it:

```
cars <- data.frame(
  car = c("Mazda RX4", "Mazda RX4 Wag", "Datsun 710", "Hornet 4 Drive", "Hornet Sportabout"),
  mpg = c(21.0, 21.0, 22.8, 21.4, 18.7),
  cyl = c(6, 6, 4, 6, 8),
  vs = c(0, 0, 1, 1, 0),
  am = c(1, 1, 1, 0, 0)
)
cars
```

```
##                 car  mpg cyl vs am
## 1         Mazda RX4 21.0   6  0  1
## 2     Mazda RX4 Wag 21.0   6  0  1
## 3        Datsun 710 22.8   4  1  1
## 4    Hornet 4 Drive 21.4   6  1  0
## 5 Hornet Sportabout 18.7   8  0  0
```

Sometimes you may say, ok, why don't we combine "vs" and "am" together and have a better view? I would say you are right! Why don't we give it a shot?!

```
unite(cars, "vs_am", c("vs","am"))
```

```
##                 car  mpg cyl vs_am
## 1         Mazda RX4 21.0   6   0_1
## 2     Mazda RX4 Wag 21.0   6   0_1
## 3        Datsun 710 22.8   4   1_1
## 4    Hornet 4 Drive 21.4   6   1_0
## 5 Hornet Sportabout 18.7   8   0_0
```

How fantastic! It's getting less messy!

This is basically it! If you still feel sort of unfamiliar with it, I highly encourage you to check out this video to get a more direct feeling about these functions and understand them better!

Take-home Message

When we are given some "untidy" data, we can always try using "tidyr" package to "tidy" the data and makes it easy to be manipulated and visualized.
Use "gather()" to gather multiple columns into key-value pairs, so that we can analyze the relationship between keys and values.
Use "spread()" to take two columns (key & value) and spread into multiple columns to make data manipulation easier.
Use "separate()" and "unite()" if you want to split one column into two or the other way around.

References

1. "Introducing tidyr." RStudio Blog, blog.rstudio.com/2014/07/22/introducing-tidyr/.
2. "Data Tidying." Data Tidying · Data Science with R, garrettgman.github.io/tidying/.
3. "Separate one column into multiple columns." Separate one column into multiple columns. - separate • tidyr, tidyr.tidyverse.org/reference/separate.html.
4. "Unite multiple columns into one." Unite multiple columns into one. - unite • tidyr, tidyr.tidyverse.org/reference/unite.html.
5. Wickham, Hadley. "Tidyr v0.4.0." Tidyr package | R Documentation, www.rdocumentation.org/packages/tidyr/versions/0.4.0.
6. "Tidyr | R Tutorial." YouTube, YouTube, 14 Apr. 2016, www.youtube.com/watch?v=RbUWwuJeUC8.
7. "Easily Tidy Data with 'spread()' and 'gather()' Functions [R package tidyr version 0.7.2]." The Comprehensive R Archive Network, Comprehensive R Archive Network (CRAN), cran.r-project.org/web/packages/tidyr/index.html.