Post2 – Data Manipulation with `tidyr`

Pingjing Wang

2017/11/27

Data Manipulation with `tidyr`

Introduction of the `tidyr`

Data frames generally occupy a central place in R analysis workflows. Although the base R functions provide many useful tools to subset, reformat, and transform data frames, the specialized packages – `tidyr` and `dplyr` offer a more succinct and faster way to manipulate and perform common data frames. Except saving typing time, the simpler syntax `tidyr` and `dplyr` provided makes scripts more readable and easier to debug. We have already learned the usages of `dplyr` in lectures, so today I will introduce how to use `tidyr` for data manipulation. `tidyr` is a new package by Hadley Wickham that will help us to organize or reshape our data in order to tidy the data and make the analysis easier. Tidy data is data that's easy to analyze and work with: we can easily change pieces of data (with `dplyr`), visualize (with `ggplot2`), and model (with R's different modeling packages). Therefore, the `tidyr` is often used in conjunction with the `dplyr`. Here comes a question. What kind of data is tidy data? There are three most important properties for a tidy data:

- Each variable is in a column
- Each row is a row
- Each value is a cell

Tidy data describes a standard way of storing data. If the data we get is tidy, we will spend less time organizing data and more time working on the analysis. Hence, learning the tidyr is really helpful for us to manipulate our data as well as the data analysis.

The `tidyr` provides four functions to help us change the layout of the data frame:

- `gather()` : converts wide data to longer format.
- `spread()` : converts longer data to wide format.
- `separate()` : splits one column into two or more columns.
- `unite()` : combines two or more columns into a single column.

I will show and explain in details how to apply these four functions for manipulating our data set throughout the following examples.

Motivation for the post

In previous lectures and labs we have already learned the main usages of `dplyr` functions such as `filter()`, `mutate()`, and `slice()`. Today I will go beyond what we've covered before and introduce four new functions `tidyr` provides:

- Use `gather()` to gather or collapse columns into rows
- Use `spread()` to spread rows into columns
- Use `separate()` to separate one column into multiple
- Use `unite()` to unite multiple columns into one

Import the Pokemon data in R

At the beginning, we need to download the data file "Pokeman.csv" to our local working repository.

```
# install and library the package "readr"
# download the data from the website https://www.kaggle.com/abcsds/pokemon/data and read into R as an csv file
library(readr)
data <- read.csv("../post2/Pokemon.csv")

# take a look at the first five rows of our data
head(data)
```

```
##   X.                  Name Type.1 Type.2 Total HP Attack Defense Sp..Atk
## 1  1             Bulbasaur  Grass Poison   318 45     49       49       65
## 2  2               Ivysaur  Grass Poison   405 60     62       63       80
## 3  3              Venusaur  Grass Poison   525 80     82       83      100
## 4  3 VenusaurMega Venusaur  Grass Poison   625 80    100      123      122
## 5  4            Charmander   Fire          309 39     52       43       60
## 6  5            Charmeleon   Fire          405 58     64       58       80
##   Sp..Def Speed Generation Legendary
## 1      65    45          1     False
## 2      80    60          1     False
## 3     100    80          1     False
## 4     120    80          1     False
## 5      50    65          1     False
## 6      65    80          1     False
```

Installing tidyr in R

After we import the Pokemon data frame into R, we need to install and library the `tidyr` package in order to get started with the above four

functions.

```
# library the package
library("tidyr")
```

```
## Warning: package 'tidyr' was built under R version 3.4.2
```

```
# For further using, install and library the package
library("dplyr")
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

Gathering columns into rows

The first function I want to introduce here is `gather()` in `tidyr`. The `gather()` function can help us gather or combine columns into rows, making the "wide" data frame longer.

The `gather()` takes the following form:

- `gather(data, key, value, ..., na.rm = FALSE, convert = FALSE)`

Inside the parameter, the first element *"data"* represents the data frame we will manipulate with. The two left columns *"key"* and *"value"* represent names of new key and value columns, as strings or symbols.

Here's an example of applying the `gather()` to our Pokemon data.

```
# combine the columns "HP", "Attack", and "Defense"
# assign our new Pokemon data frame to a new variable "gdata"
# here I just display six rows of the new data frame gdata to show what the new variable "Capacity" has and how it differs form the original data frame
gdata <- gather(data, Capacity, Subtotal, HP:Defense)
gdata <- slice(gdata, 798:803)
gdata
```

```
## # A tibble: 6 x 12
##      X.                Name  Type.1 Type.2 Total Sp..Atk Sp..Def Speed
##   <int>              <fctr>  <fctr> <fctr> <int>   <int>   <int> <int>
## 1   720 HoopaHoopa Confined Psychic  Ghost   600     150     130    70
## 2   720  HoopaHoopa Unbound Psychic   Dark   680     170     130    80
## 3   721           Volcanion    Fire  Water   600     130      90    70
## 4     1           Bulbasaur   Grass Poison   318      65      65    45
## 5     2             Ivysaur   Grass Poison   405      80      80    60
## 6     3            Venusaur   Grass Poison   525     100     100    80
## # ... with 4 more variables: Generation <int>, Legendary <fctr>,
## #   Capacity <chr>, Subtotal <int>
```

In this example, I combine the columns "HP", "Attack", and "Defense" in our original Pokemon data frame to get a new a column "Capacity" and the value this new column takes – "Subtotal" column.

In our new data frame "gdata", we have no more "HP", "Attack", and "Defense" columns, only two new columns "Capacity" and "Subtotal" that we add. In this way, we successfully convert the original wide Pokemon data frame to longer format.

To have a much stronger contrast with the original Pokemon data frame, now I will combine the columns in the original data frame from the column 6 to the column 11.

```
# combine the columns in the original data frame from column 6 to column 11
# assign our new Pokemon data frame to a new data frame "gdata2"
# here I just display six rows of the second new data frame gdata2 to show what the new variable "Capacity2" has and how it differs form the original data frame
gdata2 <- gather(data, Capacity2, Subtotal2, HP:Speed)
gdata2 <- slice(gdata2, 798:803)
gdata2
```

```
## # A tibble: 6 x 9
##       X.                Name  Type.1 Type.2 Total Generation Legendary
##    <int>              <fctr>  <fctr> <fctr> <int>      <int>    <fctr>
## 1   720 HoopaHoopa Confined Psychic  Ghost   600          6      True
## 2   720  HoopaHoopa Unbound Psychic   Dark   680          6      True
## 3   721           Volcanion    Fire  Water   600          6      True
## 4     1           Bulbasaur   Grass Poison   318          1     False
## 5     2             Ivysaur   Grass Poison   405          1     False
## 6     3            Venusaur   Grass Poison   525          1     False
## # ... with 2 more variables: Capacity2 <chr>, Subtotal2 <int>
```

After gathering, we can see that there are only 9 columns in our new data frame "gdata2" comparing with the original Pokemon data frame (13 columns). There's no doubt that we "narrow" our data frame and make it longer.

Spreading rows into columns

The second function is `spread()`. The `spread()` fucntion allows us to convert long data to wider format, which means that we can take two columns (a key-value pair) and spread them into mutiple columns.

The `spread()` function takes the following form: - `spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE)`

Inside the parameter, the first element *"data"* represents the data frame we will manipulate with. In addition to the first element, there're two variables. The first variable *"key"* represents the (mutiple) column we want to spread out, and the second variable *"value"* shows the values of the (multiple) column takes.

`spread()` is used when we have variables that form rows instead of columns. Usually when we deal with different kinds of data in daily life, we are less frequently using `spread()` instead of `gather()`.

Here's the example of applying the `spread()` to our new Pokemon data frame "gdata".

```
# spread the multiple column "Capacity" in the data frame "gdata" to get a new data frame "spdata"
spdata <- spread(gdata, Capacity, Subtotal)
spdata
```

```
## # A tibble: 6 x 12
##       X.                Name  Type.1 Type.2 Total Sp..Atk Sp..Def Speed
## * <int>              <fctr>  <fctr> <fctr> <int>   <int>   <int> <int>
## 1     1           Bulbasaur   Grass Poison   318      65      65    45
## 2     2             Ivysaur   Grass Poison   405      80      80    60
## 3     3            Venusaur   Grass Poison   525     100     100    80
## 4   720 HoopaHoopa Confined Psychic  Ghost   600     150     130    70
## 5   720  HoopaHoopa Unbound Psychic   Dark   680     170     130    80
## 6   721           Volcanion    Fire  Water   600     130      90    70
## # ... with 4 more variables: Generation <int>, Legendary <fctr>,
## #   Attack <int>, HP <int>
```

In this example, I spread out the multiple column "Capacity" that I combined in the first example, and then assign it to a new data frame "spdata", we can see that we are back to the original data frame.

Uniting multiple columns into one

The third function is `unite()`. As the name adjusts, the function takes multiple columns and paste them together into one.

The `unite()` function takes the following form:

- `unite(data, col, ..., sep = "_", remove = TRUE)`

There are only few varaibles inside the parameter of the `unite()` function. The first element *"data"* as usual means the data frame we will use. The following variable *"col"* indicates that the new(unquoted) name of column that we add, and the next one *"sep"* is the separator we want to put between values. What's more, the *"..."* in the parameter just shows the columns to unite.

`unite()` returns a copy of the data frame that includes the new column, but not the columns used to build the new column. If we would like to retain these columns, we need to add the argument `remove = FALSE`.

The following example will display the usage of `unite()` function.

```
# combine the "Name" and "Total" columns into the new one column "Description", separating by comma
# here I just display the first five rows to show how `unite()` function works
ugata <- unite(data, "Description", c(Name, Total), sep = ", ")
ugata <- head(ugata, 5)
ugata
```

```
##   X.                Description Type.1 Type.2 HP Attack Defense Sp..Atk
## 1  1             Bulbasaur, 318  Grass Poison 45     49      49      65
## 2  2               Ivysaur, 405  Grass Poison 60     62      63      80
## 3  3              Venusaur, 525  Grass Poison 80     82      83     100
## 4  3 VenusaurMega Venusaur, 625  Grass Poison 80    100     123     122
## 5  4            Charmander, 309   Fire        39     52      43      60
##   Sp..Def Speed Generation Legendary
## 1      65    45          1     False
## 2      80    60          1     False
## 3     100    80          1     False
## 4     120    80          1     False
## 5      50    65          1     False
```

In this example, after uniting the two columns “Name” and “Total” into one column “Description”, we can find out there’s no longer “Name” and “Total” columns in our new data frame, and we can get these two information just in one column.

Therefore, one of advantages of the function `unite()` is that it enables us to get the information we want just in one glance by spliting and combiningg cells to place a single, complete value in each cell.

Separating one column into multiple

The fourth function is `separate()`. It is the complement of the `unite()` function. Sometimes two variables are clumped together in one column, then the `separate()` fucntion allows us to tease them apart. In simple words, `separate()` is aimed at spliting simple columns into multiple columns.

The `separate()` function takes the following form:

- `separate(data, col, into, sep = "[^[:alnum:]]+")`

Inside the parameter, same as the previous examples, the first element *“data”* represents the data frame we are gonna deal with. Then, the second one is the column from the data set we will separate, and the third one is what we want to name in two or few new columns after we are separating the original column. The last parameter indicates what we gonna use to cut our elements when there is a comma.

Here’s an example of the `separate()` function.

```
# separate the column "Description"  in the "ugata" data frame into two columns "Name" and "Total"
sedata <- separate(ugata, Description, c("Name", "Total"), sep = ", ")
sedata
```

```
##   X.                  Name Total Type.1 Type.2 HP Attack Defense Sp..Atk
## 1  1             Bulbasaur   318  Grass Poison 45     49      49      65
## 2  2               Ivysaur   405  Grass Poison 60     62      63      80
## 3  3              Venusaur   525  Grass Poison 80     82      83     100
## 4  3 VenusaurMega Venusaur   625  Grass Poison 80    100     123     122
## 5  4            Charmander   309   Fire        39     52      43      60
##   Sp..Def Speed Generation Legendary
## 1      65    45          1     False
## 2      80    60          1     False
## 3     100    80          1     False
## 4     120    80          1     False
## 5      50    65          1     False
```

Conclusion (Take-home Message)

The `tidyr` package is like twin sister of the `dplyr` package; they are often used together. Therefore, I think this is a really good opportunity to introduce the usage of `tidyr` to you. This package is really widely used when we want to alter the latout of tabular data sets while preserving the values and relationships contained in the data sets. The operations of these two packages make it much easier to convert the untidy data into a tidy data.

Reference

1. Crucial Step Reshaping Data
2. Data manipulation with tidyr
3. Data manipulation with tidyr and dplyr
4. Dplyr-tidyr-tutorial
5. Introducing tidyr
6. Reshaping Your Data with tidyr
7. Tidyr|R Tutorial